FreeBSD™ JOURNAL
May/June 2015

MEASURE TWICE, Code Once!

- How to Install **BACULA** and Get Your First Backup Running
- **Go-based** Content Filtering Software

pfSense®

Security Gateways Without Limits

SG-2220

Perfect small security gateway for home and office
Modern, low power, no moving parts for long life
From $299

SG-2440

Scales from firewall to UTM with flexible software options
Multi-WAN, no artificial limits, easy web-based GUI
From $499

SG-4860

Built with performance and versatility in mind
Connect VPN tunnels between offices or Amazon EC2®
From $699

SG-8860

Rackmount power in a desktop footprint
Connect two systems for high availability configuration
From $899

C2758

Sturdy 1U gateway for businesses with heavy network loads
Expandable memory, storage and 1 Gigabit or 10 Gigabit Ethernet cards
From $1399 (call for partner discounts)

pfSense scales with your business at no additional cost providing unlimited firewall rules, unlimited users, unlimited VPN connections. Why pay more for limited security?

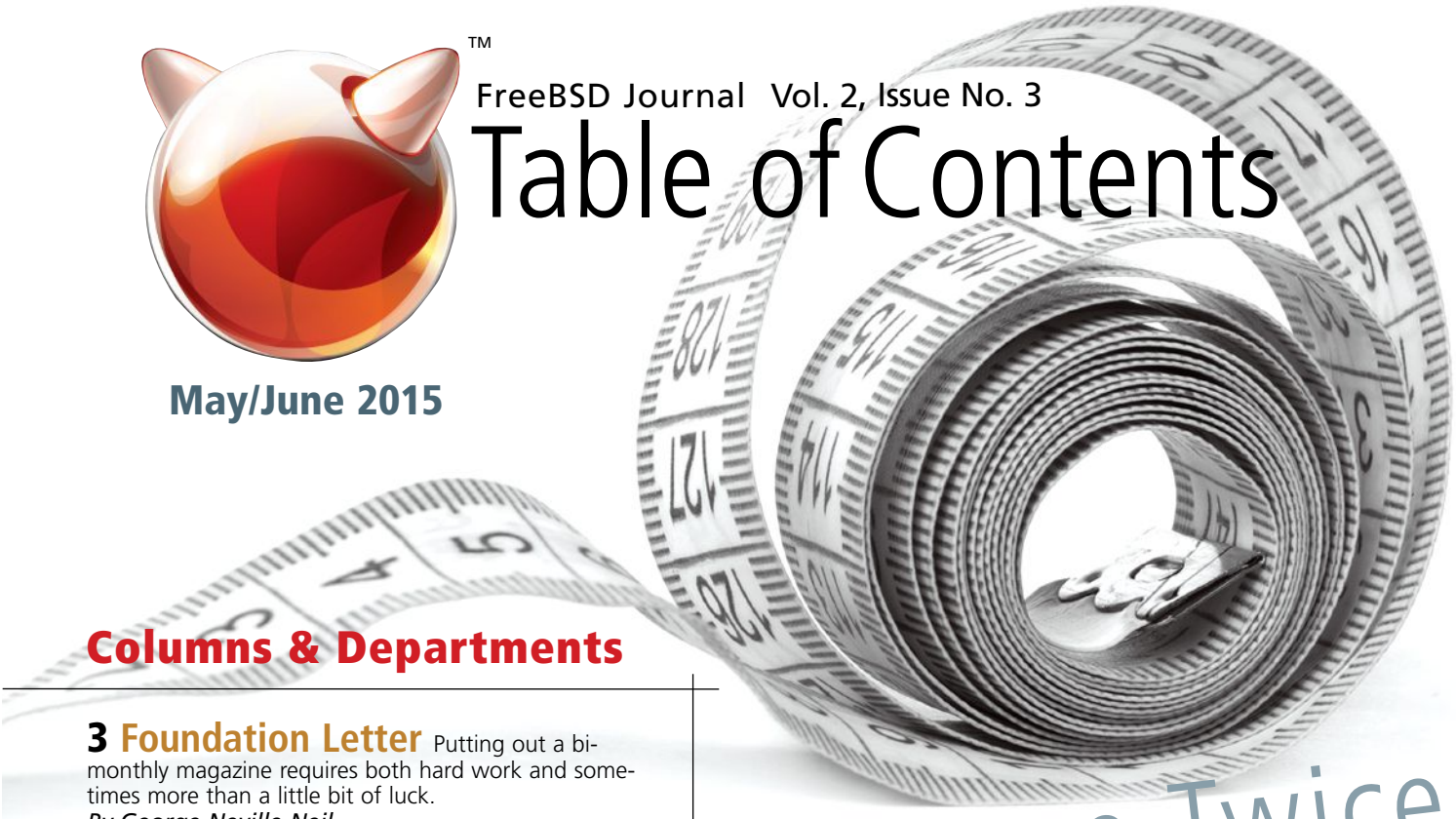
Shop now at the official pfSense store or authorized partners worldwide.

7212 McNeil Drive Suite 204 Austin, TX 78729 +1.512.646.4100

FreeBSD Journal Vol. 2, Issue No. 3

Table of Contents

May/June 2015



Columns & Departments

3 Foundation Letter Putting out a bi-monthly magazine requires both hard work and sometimes more than a little bit of luck. *By George Neville-Neil*

55 Ports Report Activity was at a high level during March and April, with important changes, especially on package building infrastructure. *By Frederic Culot*

56 Conference Report AsiaBSDCon 2015. There is a lot of laughter and stories shared with people whose names you see on emails, but who you may never have met in person. *By David Maxwell*

62 This Month in FreeBSD An interview with Julio Merino, whose interest in automated testing and code review has provided benefits to his career as well as the projects in which he is still involved. *By Dru Lavigne*

65 Events Calendar *By Dru Lavigne*

66 Book Review Peter N. M. Hansteen's *The Book of PF* is indeed a "no-nonsense guide to the OpenBSD firewall," and covers everything from the configuration basics all the way up to building highly available and redundant networks. *By Steven Kreuzer*

Interacting with the FreeBSD Project

48 FreeBSD Project Grant Proposal Submission Proposals are evaluated on technical merit, cost effectiveness, and the degree of impact the project will have on FreeBSD. *By Ed Maste*

52 FreeBSD Foundation Updates A new column by the Foundation's Executive Director will keep you informed of what the Foundation is doing to help FreeBSD. *By Deb Goodkin*

Measure Twice, CODE ONCE!

4 Network Performance Analysis for FreeBSD. This article shows developers and systems integrators at all proficiency levels how to benchmark networking systems, with specific examples drawn from the authors' experiences with the FreeBSD kernel. **By George Neville-Neil and Jim Thompson**

14 Go-based Content Filtering Software on FreeBSD. The authors look at the issues, pros, cons, and common pitfalls of developing software in Go on FreeBSD. **By Ganbold Tsagaankhuu, Esbold Unurkhaan, and Erdenebat Gantumur**

26 How to Install Bacula on FreeBSD and Get Your First Backup Running. This tutorial article teaches Bacula concepts that will help you through commonly misunderstood issues. **By Dan Langille**

FreeNAS
in an Enterprise Environment

By the time you're reading this, FreeNAS has been downloaded more than 5.5 million times. For home users, it's become an indispensable part of their daily lives, akin to the DVR. Meanwhile, all over the world, thousands of businesses universities, and government departments use FreeNAS to build effective storage solutions in myriad applications.

What you will learn...

- How TrueNAS builds off the strong points of the FreeBSD and FreeNAS operating systems
- How TrueNAS meets modern storage challenges for enterpri

The FreeNAS operating systems is freel
the public and offers thorough docu
active community, and a feature-rich
the storage environment. Based on FreeB
can share over a host of protocols (SMB,
FTP, iSCSI, etc) and features an intuitive
the ZFS file system, a plug-in system fo
much more.

Despite the massive popularity of
aren't aware of its big brother dutifu
data in some of the most demandin
environments: the proven, enterpri
professionally-supported line of a

But what makes TrueNAS differ
Well, I'm glad you asked...

Commercial Grade Suppo

When a mission critical storag
organization's whole operatio
halt. Whole community-base
free), it can't always get an
and running in a timely ma
responsiveness and expert
dedicated support team
provide that safety.

Created by the same
developed FreeNAS.

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
PO Box 20247, Boulder, CO 80308
ph: 720/207-5142 • fax: 720/222-2350
email: board@freebsdfoundation.org

Did you know that 8 is a lucky number in Asia?

And we're now at 8,800 subscribers and still climbing.
It is said that luck favors the prepared mind, and I'm inclined to agree. Putting out a bimonthly magazine requires both hard work and sometimes more than a little bit of luck.

This month we've been lucky enough to have several new authors writing for the *Journal*. Dan Langille, who many know as the leader of both BSDCan (https://www.bsdcan.org/2015/) and PGCon, has written a great article about the backup system Bacula, running on FreeBSD with ZFS. If you're not already backing up your systems, and you know that you should be, then read the Bacula piece first.

If you haven't heard someone raving about the Go programming language, then you've clearly not been reading blogs or possibly even looking at the Internet. Ganbold Tsagaankhuu, Esbold Unurkhaan and Erdenebat Gantumur take us through the how's, why's and wherefore's of filtering Internet content using the Go language on FreeBSD.

In my own work with FreeBSD, I've been focused on measuring and improving networking performance. I've taken my AsiaBSDCon 2015 (https://2015.asiabsdcon.org/) talk and paper and worked them into an article for the *Journal*, so that those who were unable to make it to Tokyo this year can get a peek at what is going on under the hood in the network stack. And speaking of AsiaBSDCon, David Maxwell has written a report for this issue that tells you all about the conference in Tokyo.

The FreeBSD Foundation (https://www.freebsdfoundation.org), which helps fund the *Journal* as well as many other activities related to FreeBSD, provides funds to developers who have a project they think would help FreeBSD, but which they do not have the money and time to pursue on their own. Ed Maste, director of projects at the Foundation, lays out just how to write a good proposal.

In columns and departments, Steven Kreuzer has written a book review of *The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall*. We're planning for book reviews to be a permanent part of the Journal so that our readers can have something else fun to read between issues. If you'd like to review a book or have a book you'd like to see us review, please let us know at editor@freebsdjournal.com. Rounding out our columns, we have the latest Ports Report from Frederic Culot, svn update from Glen Barber (our always busy release engineer), and an update on the FreeBSD Foundation by Deb Goodkin, the Foundation's executive director. Dru Lavigne has produced an interesting interview with Julio Merino in This Month in FreeBSD. Julio's journey from a student who participated in open source to a job at Google is an interesting one that spans multiple BSD projects.

As I write this, I'm preparing to head off to BSDCan (https://www.bsdcan.org/2015/), still the largest of the BSD conferences. This will be my 11th trip to Ottawa, and, I have to say that, while the city has changed quite a bit in the years that I've been going there, the great feeling from getting to meet and work face to face with the rest of the FreeBSD community hasn't. BSDCan is just one of those places where—for a week—it's all BSD, all the time. I hope to see all of you there, although I've no idea where we'll find 8,800 hotel rooms!

Sincerely, George Neville-Neil

For the *FreeBSD Journal* Editorial Board

By George V. Neville-Neil and Jim Thompson

Measure twice CODE ONCE!

Network Performance Analysis for FreeBSD

Since the birth of the Internet, the networking subsystems of any operating system have grown in complexity as the set of protocols and features supported has grown. Firewalls, Virtual Private Networking, and IPv6 are just a few of the features present in the FreeBSD kernel that were not even envisioned when the original BSD releases were developed at UC Berkeley over 30 years ago. Advances in networking hardware, with 10 Gbps NIC cards available for only a few hundred dollars, have far outstripped the speeds for which the kernel's network software was originally written. As with the increasing speed of processors over the last 30 years, systems developers and integrators have always depended on the next generation of hardware to solve the current generation's performance bottlenecks, often without resorting to any coherent form of measurement.

This article shows developers and systems integrators at all proficiency levels how to benchmark networking systems, with specific examples drawn from our experiences with the FreeBSD kernel. Common pitfalls are called out and addressed and a set of representative tests are given. A secondary outcome of

this work is a simple system for network test coordination, Conductor, which is also described. The Conductor system, as well as all the tests and results, are published, in parallel, in two open-source projects: (https://github.com/gvnn3/conductor) and (https://github.com/gvnn3/netperf).

Network Benchmarks Are Difficult

Reading over any company's published benchmarks fills a competent engineer with skepticism—and it should. Creating a broadly applicable and honest benchmark is difficult work, and many benchmarks are not meant to be unbiased. Some are quite the opposite, meant to show the author's system in the best possible light. Often a benchmark is picked and promoted because it is the first one that showed what the author thought of as good performance for the system being measured.

The technical challenges in developing a benchmark aren't limited to avoiding lying to the reader. Figuring out how to measure something reliably and repeatedly in a dynamic system requires a deep understanding of all of the system components and how interactions among them can conspire to create false measurements. A non-networked system, where the interconnections and interactions among the various components are, or should be, visible, is still complex enough to trip up many developers. A networked system is far harder to measure for several reasons, including asynchrony, visibility, and the lack of well synchronized clocks.

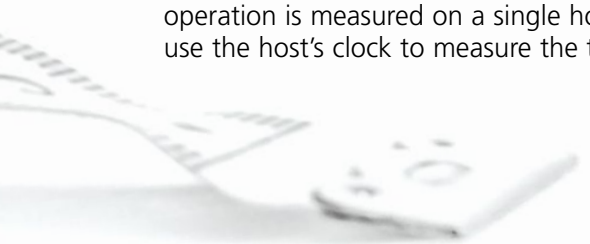
Unlike other aspects of computer systems, networks are understood to be unreliable, with higher-layer protocols providing features, such as reliability and ordering, that software running on a single host takes for granted. The acceptance of unreliability at many layers of a network stack make it difficult to measure performance from only one viewpoint, either the sender's or receiver's. To understand networking performance we must observe at several points in any system, the sender, the receiver, and, depending on the network application, any systems that exist between the two endpoints. When an operation is measured on a single host, we can use the host's clock to measure the time the operation took to complete. The difficulty of synchronizing clocks across a network means that we do not have as reliable a way to measure operational latency as if the software were on a single host. Recording the time that a packet was sent on one host, and then recording it again on another host when the packet is received, is only as reliable as the level of synchronization between two hosts. Two or more hosts can be synchronized to within a millisecond using the Network Time Protocol (NTP) or to within a microsecond using the Precision Time Protocol (PTP), but most networked systems cannot use PTP in large deployments, and, even if they could, a 10Gbps network, which is common in a modern data-center, has latencies that are measured in the low tens of microseconds, making accuracy between hosts hard to determine.

Accurate timing is just one problem facing the engineer who has to perform a network benchmark. The first problem is simply to understand the right thing to measure. Some applications are latency sensitive, some sensitive to loss, and others require large amounts of bandwidth. Working out which of these three axes matters most to the application is the primary task that must be done before a benchmark can begin. A full treatment of this and other performance measurement topics is the subject of an excellent book [3].

In this article we discuss some very basic benchmarking techniques and show how they were carried out on several systems, first looking at packet forwarding in FreeBSD [2], and then expanding to show a comparison of four firewalls. The goal of this work is not to be the definitive result for a particular measurement, but to show how we constructed our benchmarks and how the same techniques can be applied to measure and improve network performance.

Understanding Modern Hardware

Networking hardware long ago passed speeds of 1 Gbps with 10 Gbps and higher being within easy reach of even the smallest companies. While network speeds have continued to climb, processor speeds have topped out at around 3GHz and memory buses have stayed closer to processor speed increases. These speed mismatches have led to the creation of systems with architectures that attempt to split up work among many similar parts to get a



single job done. Multiple cores, memory buses, and network queues all contribute to measurement noise when benchmarking on modern hardware.

To achieve top performance, all of the system's resources, including CPU cores and caches, memory, and network queues must be in alignment. The resources are in alignment when packets with the same four-tuple of Source IP, Destination IP, Source Port, and Destination Port are always handled by the same core, cache, and queue. Why is this alignment necessary?

A high-speed network, which is currently any network that has a 10 Gbps or higher bandwidth, handles packets at such a high rate that there is very little time for the CPU to do any work with the packet. At 10 Gbps, minimum-sized packets, which contain 64 bytes, arrive at approximately 14 million per second. The system receiving these packets has 67 nanoseconds to process each one. In this time frame, a 3 GHz processor has approximately 200 cycles in which to process each packet. This seems adequate until the cost of accessing memory and caches is taken into consideration. A single cache miss can cost 32ns, which means that every time a packet arrives and the relevant data to handle that packet is not on the core to which the packet is ultimately routed, the system needs to evict data from the cache and then retrieve data from memory. Data relevant to the handling of a packet includes forwarding and routing entries, socket buffers, and all the kernel's locking data structures related to the packet. The cost of cache misses is now high enough that when working with high-speed packet flows any such process must be set to a single CPU core in order to achieve its highest performance. Allowing the scheduler to decide where the process should best be executed results in swings in the packet-forwarding rate as well as an overall lowering of performance.

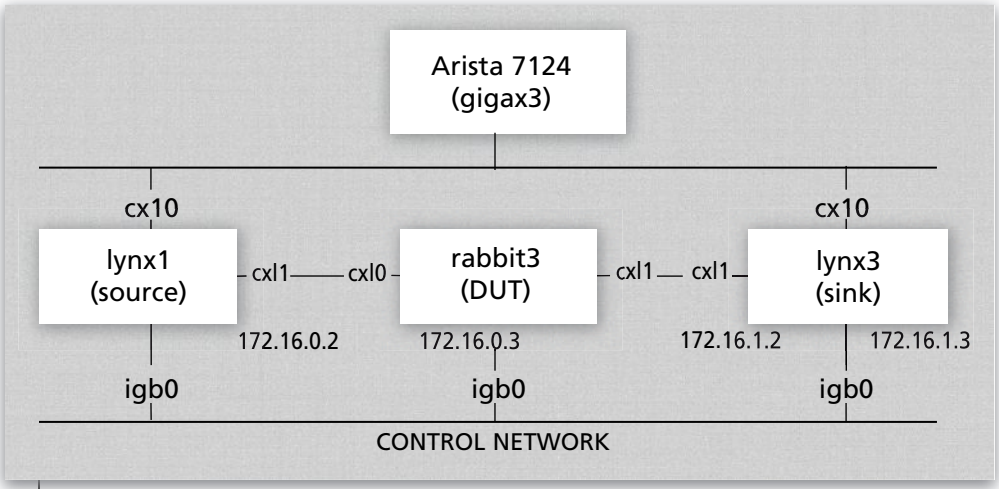


Fig. 1. Lab Setup

The only way to obtain the full performance from high-speed NICs and modern hardware is to be aware of the complete system topology. Packets must arrive and leave on a network card that is on a bus connected to the CPU that has the application running on it. If an application is moved among cores and therefore loses cache locality, performance will suffer. On FreeBSD this means that network applications must be tied to a processor core using the cpuset program, or system calls, in order to work at top speed. Not tying a networking program to a core is the most common mistake that engineers make when working with high-speed network hardware.

Multi-socket systems, those with two or more multicore CPUs, exacerbate this problem because of the high cost of inter-processor communication. Any Intel-based multi-socket system built since the advent of QPI, which includes all systems built in the last two years, connects each device, such as a network controller, to at most one CPU in the system. When a 10 Gbps NIC is plugged into the bus it will be close to only one of the CPUs. If an application handles packets on the NIC from the foreign CPU, the penalty is so high that there is no reason to bother with the extra cost of additional CPUs.

Test Automation

To undertake our work, we created a simple coordination program and protocol, Conductor, that can be used to set up, run, and reset systems without manual intervention.

The Conductor system is a small set of Python libraries that are used to coordinate various systems during testing. A typical test setup can be seen in Figure 1 where three hosts are used to test the forwarding of packets. The host lynx1 sends packets to rabbit3, the DUT, via rabbit3's cxl0 interface. The packets are forwarded by rab-

bit3 out of its cxl1 interface where they are counted by lynx3. Conductor is used to coordinate these processes. Running from a master host, zoo, which resides on the administration network and is not shown in the diagram, Conductor contacts a player program on each host and executes commands in four phases to run test scenarios in a reproducible way. The Setup phase is where device drivers are loaded, route entries are set up, and initial ping commands are used to make sure that the ARP tables are prepared for the test. Once the setup is complete, the Run phase starts up the packet generator sources and sinks. Commands can be given a timeout so that programs that do not have a built-in timeout can be forced to exit after a preprogrammed number of seconds. The Collection phase copies off the results of the tests, including log files, packet captures, and any other output related to the test. Finally, the Shutdown phase removes the routes and the drivers loaded into the kernel during the Setup phase to bring the systems back to their initial, pretest, state.

Fig. 2. Baseline TCP Performance with iperf3

```
0.00-1.00   sec  1.09   GBytes   9.41   Gbits/sec
1.00-2.00   sec  1.10   GBytes   9.41   Gbits/sec
2.00-3.00   sec  1.10   GBytes   9.41   Gbits/sec
3.00-4.00   sec  1.10   GBytes   9.41   Gbits/sec
4.00-5.00   sec  1.10   GBytes   9.41   Gbits/sec
5.00-6.00   sec  1.10   GBytes   9.42   Gbits/sec
6.00-7.00   sec  1.10   GBytes   9.41   Gbits/sec
7.00-8.00   sec  1.10   GBytes   9.41   Gbits/sec
8.00-9.00   sec  1.10   GBytes   9.41   Gbits/sec
9.00-10.00  sec  1.10   GBytes   9.41   Gbits/sec
```

Kernel Packet Forwarding

One area of the FreeBSD operating system that is ripe for both performance analysis and improvement is the forwarding of packets via the base network stack as well as through various firewalls. There are several reasons that packet forwarding is a good candidate for performance work. First, there is a simple, quantitative, measure of performance—packets per second. Secondly, the workload is both generated and measured outside of the system that is being tested, which we refer to as the Device Under Test (DUT). The external generation and reception of the packets means that the DUT is only doing the work that we care about in our testing, rather than using up cycles with overhead (See Figure 2).

Since the first versions of BSD, the operating system has had the ability to act as a router as well as an end system, forwarding packets, often as a small office router. FreeBSD also includes two well-known firewall subsystems. IPFW was originally written for FreeBSD by Luigi Rizzo [4] and has been enhanced and extended over the last 20 years. The PF firewall was developed by OpenBSD and then ported to FreeBSD. PF was not written for an SMP kernel, but was updated by Gleb Smirnoff and others to run in the FreeBSD kernel without a giant lock. IPFW and PF do not have the exact same features, but they both implement a firewall which can either pass or drop traffic between interfaces.

Before any measurements can be made of the firewall software, a baseline needs to be taken of the systems being used. We undertook several experiments to establish how quickly the kernel could forward packets without subjecting them to the work required by a firewall or the kernel itself. Our experiments were carried on three systems in the Sentex Test Lab—lynx1, lynx3, and rabbit3. Lynx1 acted as the traffic source, sending packets via rabbit3 on to lynx3. Both of the lynx machines are dual socket with 10 core E5-2680 Xeon processors clocked at 2.8 GHz. Rabbit3 is a single socket, four core, E5-2637 Xeon, running at 3 GHz. Lynx1's second 10G port was connected directly to rabbit3's first 10 G port and lynx3's second 10 G port was connected directly to rabbit3's second 10 G port as can be seen in Figure 1.

One of the easiest and most common network tests is a single stream of TCP packets. Using the iperf3 program, we established a simple baseline for TCP within our test lab. Each test was carried out over 10 seconds, and showed little or no variance in bandwidth. The output shown in Figure 2 is iperf's output, reporting bandwidth every second for the entire 10-second run.

There are many reasons why one should not use only TCP to test networking equipment. Unlike other protocols, TCP attempts to use the underlying network as efficiently as possible and it will adjust its behavior in the face of errors. While this is good for normal users, it will not show the rougher edges of a piece of networking software or equipment. As a simple baseline, it is a good place to start, but it is misleading, as this one test might lead us to believe that the system is capable of forwarding 10 Gbps in all situations, which is not the case.

Fig. 3. pkt-gen output, source side, 64 byte packets

```
827.257743 main_thread[1512] 14697768 pps (14726664 pkts in 1001966 usec)
828.259812 main_thread[1512] 14668997 pps (14699332 pkts in 1002068 usec)
829.261742 main_thread[1512] 14695277 pps (14723654 pkts in 1001931 usec)
830.263743 main_thread[1512] 14685547 pps (14714933 pkts in 1002001 usec)
```

Fig. 4. pkt-gen output, destination side, 64 byte packets

```
866.466039 main_thread[1512] 11943109 pps (11955136 pkts in 1001007 usec)
867.468024 main_thread[1512] 11946111 pps (11969824 pkts in 1001985 usec)
868.469126 main_thread[1512] 11942020 pps (11955180 pkts in 1001102 usec)
869.471027 main_thread[1512] 11939957 pps (11962655 pkts in 1001901 usec)
```

When testing systems that are responsible for forwarding packets, such as switches, routers and firewalls, we want to measure the number of packets that can be carried across the DUT. Such tests are carried out using network layer packets where the transport protocol, such as TCP or UDP, is mostly irrelevant. Before passing packets through the DUT, we established a raw baseline between two cards, connected back to back. A 10 G network can theoretically move just over 14 million 64-byte packets per second. We wanted to establish just how many packets our systems could send and receive before adding in any other software layers. Using the pkt-gen program from netmap[5] we sent 64-byte packets from lynx1 to rabbit3 over a back-to-back link. A portion of the output at the source is shown in Figure 3 and at the destination in Figure 4. We used pkt-gen because it, in turn, uses the netmap(4) driver, allowing us to send and receive packets at near line rate.

The initial test shows that minimum-sized Ethernet frames, those of 64 bytes, cannot be completely absorbed by the DUT on its 10 G interface. Running a series of packets between the source and DUT shows that packets of 128 bytes or longer are all properly received by pkt-gen. The results of a full sweep of packet sizes are shown in Table 1. By using pkt-gen and netmap we are able to write packets directly to, and read them directly from, the card without the packets going through any of the operating system's network-processing software, thus giving us a good baseline measurement of what the underlying hardware and drivers can achieve. Once we had established baseline for our cards and hosts, we then measured the ability of the operating system to forward packets between interfaces.

In the forwarding scenario, a single stream of UDP/IP packets was sent from lynx1 through rabbit3 and on to lynx3 where they were counted. No software other than the operating system was running on rabbit3, and its only job was to forward the packets between the `cxl0` and `cxl1` interfaces seen in Figure 1.

Our second set of measurements was carried out with a variety of packet sizes, including 64-, 256-, 512-, and 1,500-byte packets. Each measurement was run over 30 seconds and then the data was run through ministat to find the median forwarding rate. Jumbo frames, which are

Table 1. Card-to-Card UDP Baseline

Size	TX	RX	Stddev
64	14,644,664	11,943,632	2366
128	8,215,485	No Loss	N/A
256	4,464,323	No Loss	N/A
1024	2,332,123	No Loss	N/A
1500	820,229	No Loss	N/A

Table 2. IP Packet Forwarding

Size	TX	RX	Stddev	%Line Rate
64	14,685,502	1,069,691	165	7
128	8,215,485	1,051,849	177	14
256	4,464,323	952,227	154	21
512	2,332,123	949,432	165	41
1024	1,192,770	948,172	100	80
1500	820,229	820,215	1.44	100

packets of greater than 1,500 bytes, were not tested. Packet forwarding is measured in packets per second (PPS) which is related to raw bandwidth by multiplying the PPS by the packet size.

The results of our packet-forwarding experiment are given in Table 2. We observe that once the operating system becomes involved in the forwarding of packets, our PPS measurements drop precipitously for all but full-sized Ethernet frames. With minimum-sized frames, those of 64 bytes, the operating system kernel is only able to forward 7% of the packets that are sent to it. There are few networks that deal only in minimum-sized frames, as such frames can hold, at most, 10 bytes of data when UDP is used with IPv4. Minimum-sized frames are normally seen only as TCP ACK packets, which contain no data. Networking tests often use minimum-sized frames to show how well a piece of software or hardware handles the worst possible scenario. As the packet sizes increase, the system is eventually able to forward packets at line rate.

A packet forwarding test such as this one shows us an important aspect of networking benchmarks, that raw bandwidth is not always the best measurement of network performance. It is important to know how much overhead there is in the measurement. The smaller the packet, the more overhead there is for the amount of data transferred. A system that is forwarding minimum-sized packets, at line rate, is still forwarding less actual data than a system that is forwarding full-sized packets at line rate. A 10 Gbps network with 64-byte packets is transporting about 1.1 Gbps of data, because each 64-byte packet has 58 bytes of packet headers, 14 for Ethernet, 20 for IPv4, and 20 for TCP, and a 4 byte checksum. With 58 bytes of overhead per 64-byte packet, 6.8 Gbps of the bandwidth are being used for packet headers, along with 10 bytes left over for actual data when using UDP. With TCP there are only 6 bytes remaining for user data. When 1,500-byte packets are used, 9.4 Gbps of the bandwidth is usable data and only 354 Mbps is given to overhead.

Table 3. IP Fast Packet Forwarding

Size	TX	RX	Stddev	% Line Rate
64	14,685,502	1,093,090	634	7
128	8,215,485	1,079,852	549	14
256	4,464,323	1,273,975	141	28
512	2,332,123	1,267,776	136	54
1024	1,192,770	1,192,755	11595	100
1500	820,229	820,215	2.08	100

Fig. 5. Normal IP Packet Forwarding Callgraph

```
ether_input()
netisr_dispatch_src()
ether_nh_input()
ether_demux()
netisr_dispatch_src()
ip_input()
ip_forward()
ip_output()
ether_output()
```

Why?

One reason forwarding packets via the operating system kernel is so slow is that each packet that flows through `ip_input()` and `ip_forward()ip_output()`, the kernel's packet input, forwarding and output routines, is subjected to a very large number of checks before the packet is handed back to the outgoing network device.

The callgraph for the normal packet-forwarding path through the kernel, between `ether_input()` and `ether_output()` is shown in Figure 5. The output in Figure 5 was gathered by using `DTrace [1]` to show the stack above `ether_output()` at the point where that routine is called during packet forwarding. When a packet is forwarded via the kernel, each packet is subjected to several tests within each of the functions shown, and each of these functions adds overhead to every packet that is forwarded.

To improve the packet-forwarding performance of the system, a fast path routine was added in 2003, `ip_fastforward()`, in which many of the checks are deferred until after we know we can forward a packet. Any packet that was nominally correct and was not bound for the local machine was forwarded without being subjected to further checks. In a forwarding situation, many of the attributes that `ip_input()` and `ip_forward()` would check for, such as packet options, never occur. Deferring these expensive operations after a point at which a typical packet would be forwarded increases overall performance by lowering the per packet overhead.

Using DTrace we are able to measure the time a packet takes to traverse the system by recording the time when it arrives in `ether_input()` and leaves via `ether_output()`. A D script was

Fig. 6. Fast Packet Forwarding Callgraph

```
ether_input()
netisr_dispatch_src()
ether_nh_input()
ether_demux()
ip_fastforward()
ether_output()
```

Fig. 7. Nanoseconds per Packet, Normal Forwarding

```
value-------Distribution----------count
512                                     0
1024                              1414505
2048                                35478
4096                                  481
8192                                    0
```

Fig. 8. Nanoseconds per Packet, Fast Path

```
value--------Distribution---------count
512                                     0
1024                              1721837
2048                                41287
4096                                  490
8192                                    0
```

Fig. 9. OpenBSD pf Rule-set

```
set  limit states 100000000
match out on ix0 from 198.18.0.0/24 to any nat-to ix0
pass in quick all keep state
pass out quick all keep state
```

run for 30 seconds to record the time between these two calls. During those 30 seconds the DUT was subjected to a stream of 512-byte, UDP/IP packets, which is the packet size that showed a large difference between the normal and fast-forwarding cases in Tables 2 and 3.

Figure 7 shows a histogram of the times between `ether_input()` and `ether_output()` without fast forwarding, the baseline case. Figure 8 shows the same script with the same packet stream with fast forwarding turned on. Note that far more of the times fall into the 1024 bucket than into any of the others. The lowered per-packet processing time leads directly to the higher PPS shown in Table 3.

A Test of Firewalls

Several sets of measurements were carried out to compare various firewalling solutions. Four software stacks were tested: FreeBSD 11-CURRENT, pfSense 2.2 (based on FreeBSD 10.1), OpenBSD 5.6, and CentOS 7. FreeBSD (http://www.freebsd.org), pfSense (http://www.pfsense.org), and OpenBSD (http://www.openbsd.org) all used the pf packet filter, and CentOS 7 used Linux's iptables. All of the measurements were carried out on different hardware than that used in the Kernel Packet Forwarding section. For this set of tests, hardware that is normally used for corporate firewalls was chosen to give a more realistic example of what can be achieved with open-source firewall software. The DUTs are C2758, 1U systems, equipped with a single, 8 core, 2.4 GHz Atom processor, 8 G of RAM and an Intel 82599 based 10 G network interface. The source and sink systems were Xeon HC boxes X5680 @ 3.33GHz with Intel 82599 based 10G network interfaces running with Linux and DPDKs pktgen on the source, and FreeBSD 11-CURRENT (NODEBUG kernel) on the sink. A slower box was intentionally chosen for the DUT to ensure that the source could outperform it and place it under sufficient stress during testing.

The DUT was placed between the source and sink in back-to-back fashion, similar to what is shown in Figure 1 with lynx1, rabbit3, and lynx3. Operating-system kernels were chosen such that they did not have expensive debugging overhead included in them. In particular, on FreeBSD 11, which is the head of the tree under active development, the GENERIC-NODEBUG kernel was used so that features that help with debugging SMP systems, such as the WITNESS system for tracking lock order reversals, are turned off. A series of packets was then sent through each DUT and measured at the sink host (Figure 8).

The rule sets used for each test were created to be nearly identical in function. The OpenBSD rule set is shown in Figure 9, while the pfSense and FreeBSD rule sets, which are identical, are shown in Figure 10. The Linux rule set is shown in Figure 11. Each rule set sets up the DUT to perform network address translation on packets that are transiting the system via the 10 G interface. Because NAT requires the system to modify every packet that passes through the DUT, it is a good test of the performance of the software stack.

The first set of measurements tested the DUT's ability to forward packets using only a single core of the eight available in the host with the NAT filtering disabled. Forwarding packets without NAT gives us a baseline to measure against in the following test. The results shown in Table 4 give an idea of the per-core, packet-forwarding power of

the DUT. In this set of tests, pfSense fared best and Linux worst, with pfSense forwarding more than twice as many packets as Linux, and twice as many as a standard installation of FreeBSD 11. pfSense is built as a firewall, rather than as a general purpose operating system, and we can see here that it is well tuned to its job.

With the filtering turned on, the packet rates drop, as seen in Table 5, as each packet filter is inspecting and modifying each packet that it sees. We see that pfSense remains ahead of all the other systems and that FreeBSD 11 drops to last, just behind Linux.

A single core may be an interesting way to establish a scaling target— that is, in a perfect world, if a single core can forward N packets, then M cores can forward M * N packets. In real-world situations this is not the case due to numerous factors, including scheduling overhead, cache pollution, interrupt routing and others which will be covered in future work. In Table 6 we see what each system can forward when using all the available CPU cores with filtering turned off. With multiple cores we see that Linux and pfSense shoot far ahead of OpenBSD and that FreeBSD also lags behind the two leaders.

Looking at Table 6, we are struck by the fact that OpenBSD does not show any increase in performance. The reason for this poor showing is that OpenBSD remains a single-threaded kernel, with a single lock protecting all kernel data structures, and therefore cannot take advantage of the extra cores in the system. The final set of firewall tests shows the performance of each system while filtering packets using all available cores. As before, OpenBSD does not gain any advantage from multiple cores and even falls slightly behind its single-core performance, possibly due to caching effects with multiple cores, and caches come into play. CentOS increases its performance over filtering on a single core by roughly the same amount as it increased when not filtering packets 4.7x and 4.8x speed-up respectively. The pfSense system increases its packet rate considerably more when filtering than when not filtering (1.8x and 2.2x).

A full comparison of all the results can be seen in Figure 12. The top of each bar gives the measured packets per second while the labels along the X-axis indicate whether the measurement was taken with filtering turned on or off, as well as whether the DUT was using one or more cores.

Fig. 10. FreeBSD and pfSense Rule-set

```
set limit states 100000000
nat on ix0 from 198.18.0.0/24 to any -> ix0
pass in quick all keep state
pass out quick all keep state
```

Fig. 11. Linux iptables

```
iptables -t nat -A POSTROUTING -j MASQUERADE -s 198.18.0.0/24 -o enp4s0f0
iptables -t filter -A FORWARD -j ACCEPT -m conntrack --ctstate ESTABLISHED,RELATED
iptables -t filter -A FORWARD -j ACCEPT
```

Table 4. Single-Core Performance w/o Filtering

OS Version	PPS	StdDev
pfSense 2.2	494,224	1944
OpenBSD 5.6	360,147	1162
FreeBSD 11-CURRENT	249,464	498
CentOS 7	198,239	172

Table 5. Single-Core Performance with Filtering

OS Version	PPS	StdDev
pfSense 2.2	228,558	1440
OpenBSD 5.6	187,523	78
CentOS 7	139,797	95
FreeBSD 11-CURRENT	131,795	229

Table 6. Multicore Performance w/o Filtering

OS Version	Multicore	Single Core	Speedup
CentOS 7	945,807	198,239	4.7x
pfSense 2.2	920,415	494,224	1.8x
FreeBSD 11-CURRENT	684,721	249,464	2.4x
OpenBSD 5.6	361,253	360,147	N/A

Conclusions

As was stated at the outset, benchmarks are difficult to design and to evaluate. Modern hardware, with multiple cores, differing cache, memory, and device hierarchies, makes the process of getting reliable numbers from a set of benchmarks that much more difficult. Furthermore, tools to

Table 7 Multicore Performance with Filtering			
OS Version	Multicore	Single Core	Speedup
CentOS 7	681,980	139,797	4.8x
pfSense 2.2	505,417	228,558	2.2x
FreeBSD 11-CURRENT	282,163	131,795	2.1x
OpenBSD 5.6	186,865	187,523	N/A

extract performance information from computer systems, such as on-chip, hardware-performance monitoring counters, Intel's pcm and VTune tools, and others, provide either too much or too little data and all suffer from a probe effect whereby their use unduly influences the DUT.

In this article we chose a very simple workload—packet forwarding—for several reasons. The first is that the measurement tools are completely outside of the DUT, and reside on boxes where we do not care about the performance of the system so long as it is capable of generating and absorbing the required number of network packets. Keeping the tools outside the DUT thereby reduces the number of variables we have to control for on the system. Secondly, the measurement, packets per second, is easy to understand from a quantitative perspective—more is better, which makes comparing various systems, such as firewalls far easier.

We could have, and may yet, measure the system resources, such as CPU utilization, on the DUT, but we did not, because we were not concerned with whether the DUT was using 10% or 90% of its CPU; we only cared about how many packets it could forward.

As has been repeatedly pointed out over the last 10 years, taking full advantage of multicore systems matters, and it will only matter more as core counts increase and CPU frequencies do not. The lesson of OpenBSD should not be ignored.

The addition of distributed-systems coordination software, using Conductor, has saved quite a bit of time and allowed us to run more experiments more frequently than we could have by hand. Most distributed systems testing involves tedious setup work to get the sources, sinks, and DUTs configured and then to collect the data. Having an automated system to aid in our efforts was a clear gain.

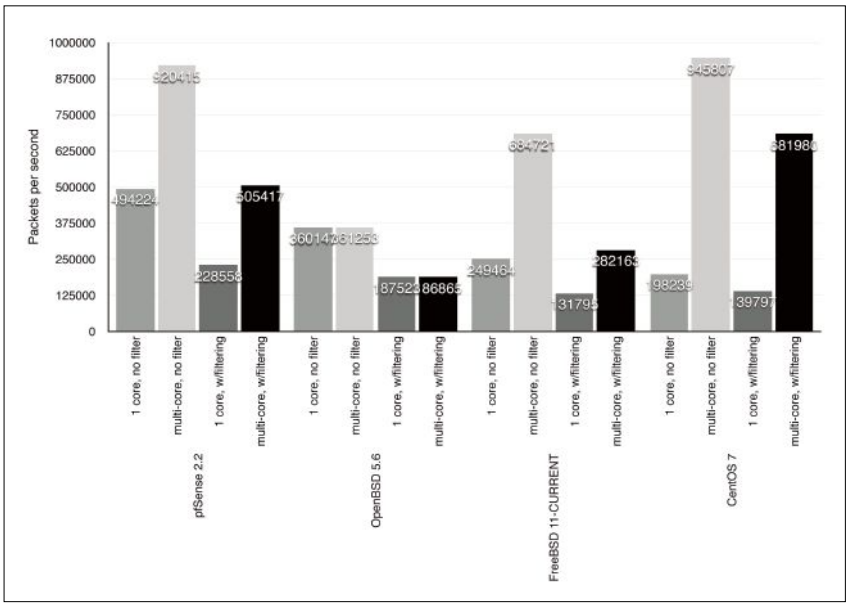


Fig. 12. Complete PF Comparison

Status and Future Plans

Both the netperf and Conductor projects are current and ongoing. The relevant code repositories track both the work on automation as well as new test scenarios and results. It is our intention to continue to make periodic surveys of network performance on the BSDs in order to see how they change and improve over time.

Acknowledgments

This work has been directly supported by Rubicon Communications (Netgate), which has paid for the work involved to create the Conductor system as well as the time taken to compare and improve IPFW and PF in FreeBSD. We would especially like to thank Matthew Smith of Netgate for his help with the firewall testing. The FreeBSD Foundation and Sentex Communications support a test cluster that includes the high-performance networking equipment used to carry out several of the tests described in this article.

Bibliograpy/References

[1] Cantrill, B.; Shapiro, M.; and Leventhal, A. *Dynamic Instrumentation of Production Systems*. In USENIX Annual Technical Conference, page 137. (June 2007)

[2] McKusick, M.; Neville-Neil, G.; and Watson, R. *The Design and Implementation of the FreeBSD Operating System*. Pearson Education. (2014)

[3] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Interscience. (1991)

[4] Rizzo, L. *Dummynet: A Simple Approach to the Evaluation of Network Protocols*. SIGCOMM Computer Communication Review, 27(1):31–41. (January 1997)

[5] Rizzo, L. *netmap: A Novel Framework for Fast Packet I/O*. Presented as part of the 2012 USENIX Annual Technical Conference, pages 101–112, Boston, Massachusetts. (2012)

Jim Thompson has been noodling around the UNIX world for far too long a time. He knows he started with BSD Unix Release 4.0 on a Vax 11/780 in 1980. He still thinks "echo 'This is not a pipe.' | cat - > /dev/tty" is funny. He submitted his first patch to a Free Software project in 1985 for a port of GNU Emacs to a Convex vector supercomputer. Jim refuses to divulge his qualifications and may, in fact, have none at all. He lives in a fortified compound near Austin, Texas, with his wife, Jamie, and son, Hunter S. Thompson. His email address is jim@netgate.com.

George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, networking and time protocols. He is the coauthor with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System*. For over 10 years he has been the columnist better known as Kode Vicious. He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler and currently lives in New York City.

This paper was published in a different form by AsiaBSDCon 2015 https://2015.asiabsdcon.org/.

BY GANBOLD TSAGAANKHUU, ESBOLD UNURKHAAN, ERDENEBAT GANTUMUR

Content Filtering Software on FreeBSD

Go is a new programming language—new when compared to many other programming languages like C, C++, Java, etc. It has many practical features and is in many cases more productive than some of the others. On the other hand, FreeBSD has been around for a very long time and proven to be the most reliable and most powerful operating system available today. In this article, we discuss the issues, pros, cons, and common pitfalls of developing software in Go on FreeBSD. We chose content-filtering software for this purpose and called our project Shuultuur. Shuultuur is a Mongolian word that means "filter" in English.

In everyday life, we are witnessing how the modern world is moving towards the Internet of Things and how the connected world is expanding quickly. This shift made us look into existing problems from a different perspective. For us, one of these problems was content filtering, and with that in mind we began exploring the possibilities.

There are countless numbers of open-source projects that are backed by various communities. However, some projects didn't evolve well throughout their lifetimes, and it became difficult to improve the existing code base. Since our content-filtering project grew out of specific needs, we decided to develop it from scratch. To save years of development time, we needed to be productive to match up with mature content-filtering software's features and performance, and then take it further.

RATIONALE

• *Why content filter?* First of all, in regard to the main objective of content filtering, the common understanding is to have some sort of control over unwanted content. These types of solutions are widely used in enterprises to enforce computer security policies. Public organizations such as libraries, schools, etc., use content filters to protect children from content inappropriate for their age—e.g., adult, violence, drugs, etc.

• *Programming language choice.* One of the questions was which programming language to use? Essentially, we were looking for a programming language that is fast, lightweight, easy to prototype, and that requires relatively minimal effort to produce and maintain production-quality code. Therefore, we preferred a statically typed,

compiled language with strong type system. Although there is always a tradeoff, our need to achieve our goal faster made our stated characteristics essential.

Go was officially launched at Google five years ago (1). It is a compiled, statically typed, garbage-collected, unconventionally object-oriented general-purpose system programming language. Go produces native binaries, has a very fast compilation time, and is designed with concurrency in mind. Therefore, since Go's characteristics matched with our initial requirements, we looked closely into it along with other candidates and started experimenting. Performance of Go's native binaries was somewhat comparable to good old low-level C and tremendously better than interpreted languages (5). After extensive research and trials, we concluded that the Go programming language was the language best suited for our goal.

Go did not need an additional library to deal with concurrency as it is already part of the programming language features, and Go has strong support for multiprocessing. In addition, Go is a more productive language compared to C and includes multiple, useful built-in data structures such as maps (23) and slices (24). Especially when dealing with concurrency, many advanced practical solutions can be easily employed in modern hardware using Go-specific features such as goroutines (21) and channels. A goroutine is a function executing concurrently with other goroutines in the same address space. It is lightweight and communicates with other goroutines via channels (22). Because Go is a very simple, garbage-collected and statically typed language, source code can be written with fewer errors, and presumably with fewer bugs. Furthermore, it is relatively easy to profile for speed and memory leaks, which is handy when working on production source code. In terms of syntax, it is loosely derived from C and influenced by other languages such as Python (3). Also, Go has an extensive number of libraries (2), and finally, Go is a BSD-licensed completely open-source language (4).

In the context of content filtering, to accurately detect the meaning of a sentence is a hard task for an automated tool. As we know, humans cannot detect the real meaning of sentences without detailed information. However, in the real world, content filters try to classify web content based on string-matching techniques into bad content, which should be blocked, or good content, which should be allowed (6). In most languages, the exact string-matching technique (pattern matching) may demand high processing power (7). For this reason, we chose to develop content-filtering software to take advantage of Go's performance.

• ***Why FreeBSD as a platform?*** We chose FreeBSD OS as a main development and testing platform mainly because:

- ***It is one*** of the most powerful, mature, and stable operating systems, with a complete, reliable, self-consistent distribution.
- ***FreeBSD's*** networking stack is solid and fast (8).
- ***One of the*** advantages of choosing FreeBSD is its port and package system, which makes it easy to install and deploy the necessary applications and software.
- ***Handy tools*** such as NanoBSD exist, which can be used to make custom FreeBSD image easily.
- ***Finally,*** we love FreeBSD.

• ***We also used*** the following open-source software:

- ***goproxy*** provides a customizable HTTP proxy library for Go. It supports regular HTTP proxy, HTTPS through CONNECT, and "hijacking" HTTPS connection using a "Man-in-the-Middle" style attack. The intent of the proxy is to be usable with a reasonable amount of traffic, yet customizable and programmable (9).
- ***gcvis*** visualizes Go program, gctrace data in real time (10).
- ***profile*** is a simple profiling support package for Go (11).
- ***go-nude*** is nudity detection with Go (12).
- ***xxhash-go*** is a go wrapper for C xxhash—an extremely fast hash algorithm, working at speeds close to RAM limits (13).
- ***powerwalk*** is a Go package for walking files and concurrently calling user code to handle each file (14).
- ***redigo*** is a Go client for the Redis database (15).
- ***Redis*** is an open-source, BSD-licensed, advanced key-value cache and store (16).

CHALLENGES

We faced several problems during development:

The Shallalist blacklist contains more than 1.8 million URL/Domain entries. Storing them in memory was challenging and initially we stored the URL/Domain entries in Redis in the following way:

```
...
// Store URL/Domains as key and category as value
conn.Do("SET", urls_or_domain, category)
...
```

This was not effective in terms of memory utilization and performance. After a bit of research, we found a way to reduce it to around 4,100 hash keys. We used Stephane Bunel's xxhash-go to compute a hash from each URL/Domain and sliced it, and then stored those slices in Redis similar to:

```
...
// use xxhash to get checksum from URL/Domain
blob := []byte(url_or_domain)
h32g := xxh.GoChecksum32(blob)

/*
 * Store it as hash in Redis in following way:
 *     key   = 0xXXXX (first half of URL/Domain),
 *     field = XXXX   (second half of URL/Domain),
 *     value = category
 */
hash_str := fmt.Sprintf("0x%08x", h32g)
key      := hash_str[0:6]
value    := hash_str[6:]
conn.Do("HSET", key, value, category)
...
```

Banned and weighted phrase-lookup problems: Originally they were stored in Redis, and accessing them in a loop was slow and inefficient. We improved this using a graph and map. Every word that exists in the phrase lists (banned, weighted, etc.) is an edge of the graph and it should be unique in the same category. For example, we have banned phrases such as "*sex woman*," "*sex man*" and "*mature sex*." Shuultuur creates four Edges such as "*sex*," "*woman*," "*man*," "*mature*" and Vertices. The Edges and their related Vertices are stored in the map because of programming efficiencies. Go provides a built-in map type that implements a hash table. In addition, we have replaced a regular expression-based search algorithm with the Boyer Moore search algorithm implemented in Go.

Reading HTTP response bodies into strings makes the heap memory usage grow very large due to lots of allocations, especially when the rate of connections per second is high. Ideally, this should be processed using a streaming parser utilizing the io.Reader interface. Also, limiting the connection rate on incoming requests could be an option. We have optimized and improved it by doing some CPU and memory profil-

ing (19). This is done by enabling memory profiling in Shuultuur and we have used Go's built-in profiler pprof. The following report shows memory allocations during the initial stage of development:

```
# go tool pprof --alloc_space ./shuultuur_mem /tmp/profile228392328/mem.pprof
Adjusting heap profiles for 1-in-4096 sampling rate
Welcome to pprof! For help, type 'help'.
(pprof) top15
Total: 11793.7 MB
3557.7      30.2%      30.2%     3557.7      30.2%   runtime.convT2E
1212.1      10.3%      40.4%     1212.1      10.3%   container/list.(*List).insertValue
 832.3       7.1%      47.5%     2434.8      20.6%   github.com/garyburd/redigo/
                                                      redis.(*conn).readReply
 807.9       6.9%      54.4%     1874.6      15.9%   github.com/garyburd/redigo/redis.(*Pool).Get
 673.8       5.7%      60.1%      673.8       5.7%   github.com/garyburd/redigo/redis.Strings
 544.5       4.6%      64.7%      549.4       4.7%   main.regexBannedWordsGo
 521.1       4.4%      69.1%      521.1       4.4%   bufio.NewReaderSize
 490.9       4.2%      73.3%      490.9       4.2%   bufio.NewWriter
 438.2       3.7%      77.0%      438.2       3.7%   runtime.convT2I
 369.8       3.1%      80.1%     7622.9      64.6%   main.workerWeighted
 255.0       2.2%      82.3%      255.9       2.2%   main.regexWeightedWordsGo
 235.5       2.0%      84.3%      235.5       2.0%   bytes.makeSlice
 229.9       1.9%      86.2%      397.1       3.4%   io.Copy
 168.3       1.4%      87.6%      168.3       1.4%   github.com/garyburd/redigo/redis.String
 162.6       1.4%      89.0%     4048.9      34.3%   main.getHkeysLen
(pprof)
```

In this initial report you can see lots of allocations in main.workerWeighted and main.getHkeysLen. Those functions were used for searching banned and weighted phrases using Redis. We improved Shuultuur by removing those functions, did some code-level optimizations and introduced a better algorithm. Here is the report generated by the same command after the previously mentioned improvements were done and we think that there is still some room for further improvements.

```
# go tool pprof --alloc_space ./shuultuur /tmp/profile287823990/mem.pprof
Adjusting heap profiles for 1-in-4096 sampling rate
Welcome to pprof!  For help, type 'help'.
(pprof) top30
Total: 2156.3 MB
    596.9      27.7%      27.7%    1066.4      49.5%   io.Copy
    406.3      18.8%      46.5%     406.3      18.8%   compress/flate.NewReader
    177.3       8.2%      54.7%     177.4       8.2%   bytes.makeSlice
    113.5       5.3%      60.0%     115.4       5.4%   code.google.com/p/go.net/html.
                                                        (*Tokenizer).Token
     78.3       3.6%      63.6%      78.3       3.6%   code.google.com/p/go.net/html.(*parser).addText
     68.4       3.2%      66.8%      68.4       3.2%   strings.Map
…
     41.7       1.9%      77.2%      41.7       1.9%   concatstring
     37.7       1.7%      78.9%     736.6      34.2%   main.ProcessResp
     27.9       1.3%      80.2%      27.9       1.3%   makemap_c
…
     12.8       0.6%      91.8%      44.5       2.1%   bitbucket.org/hooray-976/shuultuur/db.GraphBuild
     12.5       0.6%      92.4%      12.5       0.6%   strings.genSplit
     10.7       0.5%      92.9%     595.5      27.6%   main.getContentFromHtml
```

The following is a top report that shows CPU usage in the beginning of development and it was very high.

```
…
lastpid:  1189;  load averages:  7.30,  2.42,  0.93 up 0+00:30:51  14:57:41
61 processes:  1 running, 60 sleeping
CPU: 20.5% user,  0.0% nice, 42.0% system,  6.6% interrupt, 31.0% idle
Mem: 104M Active, 63M Inact, 225M Wired, 234M Buf, 7502M Free
Swap: 16G Total, 16G Free

  PID   USERNAME    THR  PRI  NICE    SIZE    RES   STATE   C    TIME     WCPU   COMMAND
 1131   tsgan        22   52     0    182M 46196K   uwait   4    9:29  685.50%   shuultuur
  900   redis         3   52     0  69952K 42512K   uwait   6    1:11   88.48%   redis-server
 1130   tsgan         6   20     0  37856K  9084K  piperd   1    0:01    0.00%   gcvis
  918   tsgan         1   20     0  72136K  5832K  select   5    0:00    0.00%   sshd
  889   squid         1   20     0  70952K 16412K  kqread   5    0:00    0.00%   squid
 1049   tsgan         1   20     0  38388K  5168K  select  11    0:00    0.00%   ssh
  998   tsgan         1   20     0  72136K  5904K  select   9    0:00    0.00%   sshd
  919   tsgan         1   20     0  17564K  3528K   pause   2    0:00    0.00%   csh
  868   root          1   20     0  22256K  3284K  select  11    0:00    0.00%   ntpd
```

As you can see, the following top report shows much less CPU usage after optimizing the banned and weighted phrase search.

```
lastpid:  1253;  load averages:  0.15,  0.31,  0.32 up 0+00:55:22  11:55:42
45 processes:  1 running, 44 sleeping
CPU:  1.4% user,  0.0% nice,  0.0% system,  0.0% interrupt, 98.6% idle
Mem: 96M Active, 72M Inact, 279M Wired, 310M Buf, 7445M Free
Swap: 16G Total, 16G Free

  PID   USERNAME    THR  PRI  NICE    SIZE    RES   STATE   C    TIME     WCPU   COMMAND
 1183   root        17   20     0    142M 37348K   uwait   0    7:28   14.31%   shuultuur
  896   redis         3   52     0  78144K 62896K   uwait   3    0:52    0.00%   redis-server
 1182   root          6   20     0  45048K 16840K   uwait   9    0:16    0.00%   gcvis
  993   tsgan         1   20     0  72136K  6744K  select   9    0:06    0.00%   sshd
 1187   tsgan         1   20     0   9948K  1600K  kqread  10    0:03    0.00%   tail
 1091   tsgan         1   20     0  16596K  2548K    CPU8   8    0:02    0.00%   top
 1204   tsgan         1   20     0  38388K  5164K  select   5    0:00    0.00%   ssh
 1196   tsgan         1   20     0  72136K  5904K  select   1    0:00    0.00%   sshd
  885   squid         1   20     0  70952K 16384K  kqread   0    0:00    0.00%   squid
…
```

Figure 1 shows memory usage when the program was not optimized.
Figure 2 shows memory usage after some optimizations.

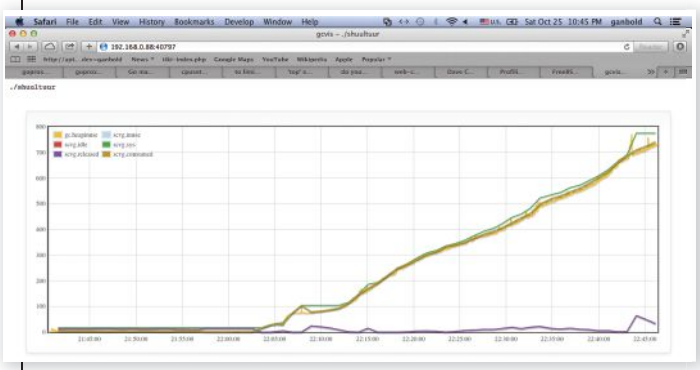


Fig. 1: Memory usage before optimization

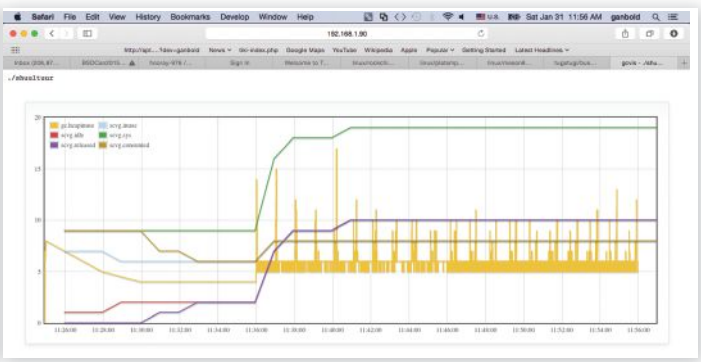


Fig. 2: Memory usage after optimization

We have implemented a number of other improvements such as learning URL/Domains to not check banned and weighted phrases every time in HTTP response bodies. The learned mode feature was added in the following way:

```
     // Learn and store this URL to redisdb temporarily
     // usexxhash to get checksum from URL/Domain
     blob1 := []byte(requrl)
     h32g := xxh.GoChecksum32(blob1)

     // key = 0xXXXXXXXX for expire_time seconds, 1 for BLOCK,
     // 2 for PASS
     key := fmt.Sprintf("%s0x%08x", policy, h32g)

     //SET key value [EX seconds] [PX milliseconds] [NX|XX]
     db.Exec("SET", key, BLOCK, "EX", EXPIRE_TIME, "NX")
…
```

Rate limiting on incoming requests was done again utilizing Redis like:

```
…
// Will set this via config file
limit := 10

// Increment counter for request
// (this will create a new key if it does not exist)
current, err := redis.Int(db.Incr(url_path + remote_addr))
…

// Check if the returned counter exceeds our limit
if current > limit {
     fmt.Println(">>> Too many requests -", url_path + remote_addr)
     response := goproxy.NewResponse(request,
     goproxy.ContentTypeHtml, 429, "Too many requests!")

     return request, response

} else if current == 1 {
     fmt.Println(">>> SET counter for:", url_path + remote_addr)

     // Set expiry on fresh counter for the given url_path and remote
     // address
     db.Exec("SETEX", url_path + remote_addr, 1, 1)
}
…
```

Another improvement was to add a possibility of limiting the listener to a specified number of simultaneous connections:

```
…
type Server struct {
      *http.Server

      // Limit the number of outstanding requests
      ListenLimit int
}

…
func (srv *Server) ListenAndServe() error {
```

CONTINUES

CONTINUED

```
        …
        l, err := net.Listen("tcp", addr)
        l = netutil.LimitListener(l, srv.ListenLimit)
        return srv.Serve(l)
}

…
if LISTEN_LIMIT_ENABLE == 1 {
        srv := &Server {
                ListenLimit: LISTEN_LIMIT,
                Server: &http.Server{Addr: ":8080", Handler: proxy},
        }
        log.Fatal(srv.ListenAndServe())
} else {
        log.Fatal(http.ListenAndServe(":8080", proxy))
}
…
```

Slow image filtering on HTTP response is temporarily disabled until we find a proper solution.

One last major issue could be related to the high number of goroutines under heavy load, which results in high CPU and memory usage. Currently we are investigating the issue (17).

BENCHMARK RESULTS

Case 1

To compare performance of our implementation with existing solutions, we used Dansguardian-2.12.0.3 and tested it in the same environment. We know that Dansguardian is usually used with squid, so we used squid version 3.4.8_2 in our test. Our content-filtering software, Shuultuur, was written in Go 1.3.2 on FreeBSD/amd64. We used the same server for this performance test comparison and the Internet link speed was 5Mbps. The server's technical specifications are listed below:

- CPU - Intel(R)Xeon(R) X5670 2.93GHz
- Memory - 8192MB
- FreeBSD/SMP -12 CPUs (package(s) x 6 core(s) x 2 SMT threads)

We used FreeBSD 9.2-RELEASE and /etc/sysctl.conf includes the following:

```
kern.ipc.somaxconn = 27737
kern.maxfiles = 123280
kern.maxfilesperproc = 110950
kern.ipc.maxsockets = 85600
kern.ipc.nmbclusters = 262144
net.inet.tcp.maxtcptw = 47120
```

We also had to change tcp-backlog setting to high value in the Redis config file. Furthermore, we performed HTTP load test using http_load-14aug2014 (parallel and rate test) (18) for both Dansguardian and Shuultuur. In http_load test, we used following URLs:

- http://fxr.watson.org/fxr/source/arm/lpc/lpc_dmac.c
- http://www.news.mn/news.shtml
- http://mongolian-it.blogspot.com/
- http://www.patrick-wied.at/static/nudejs/demo/
- http://news.gogo.mn/
- http://www.amazon.com/
- http://edition.cnn.com/?refresh=1
- http://www.uefa.com/
- http://www.tmall.com/

- http://www.reddit.com/r/aww.json
- http://nginx.com
- http://www.yahoo.com
- http://slashdot.org/?nobeta=1
- http://www.ikon.mn
- http://www.gutenberg.org
- http://en.wikipedia.org/wiki/BDSM
- http://www3.nd.edu/~dpettifo/tutorials/testBAD.html
- http://penthouse.com/#cover_new?{}
- http://www.playboy.com
- http://www.bbc.com/earth/story/20141020-chicks-tumble-of-terror-filmed
- http://173.244.215.173/go/indexb.html
- http://breakingtoonsluts.tumblr.com/

Some of the above URLs are listed in the Shallalist blacklist, some URLs contain phrases which are in the banned and weighted phrase lists, some URLs have lots of content and javascript, and the rest of the URLs are chosen with no particular reason. The following test commands were used for HTTP load tests:

```
./http_load -proxy 172.16.2.1:8080 -parallel 10 -seconds 600 urls
./http_load -proxy 172.16.2.1:8080 -rate 10 -jitter -seconds 600 urls
```

The option `-parallel` in the first command indicates the number of concurrent connections to be established and maintained, the `-rate` option in the second command controls the number of requests sent out per second, the `—jitter` option varies the rate by about 10%, and the `-seconds` option `indicates` the number of seconds to run the test.

Table 1 shows the comparison of http_load parallel and rate test results.

NO	RESULT NAMES		PARALLEL TEST		RATE TEST	
			SHUULTUUR	DANSGUARDIAN	SHUULTUUR	DANSGUARDIAN
1	Fetches		17654	4298	5991	5389
2	Max parallel		10	10	95	606
3	Mean bytes/ connection		79213.8	94820.7	72666.3	27437.2
4	Fetches/sec		29.4233	7.16333	9.985	8.98166
5	Msecs/ connect		0.189717 mean, 13.855 max, 0.088 min	0.184428 mean, 0.485 max, 0.088 min	0.177924 mean, 2.037 max, 0.106 min	0.345489 mean, 0.782 max, 0.12 min
6	Msecs/ first-response		229.182 mean, 5114.55 max, 8.049 min	1374.9 mean, 40977.9 max, 0.779 min	1189.41 mean, 59271.7 max, 11.144 min	26442.1 mean, 59925.3 max, 3.322 min
7	Timeouts		-	-	107	3432
8	Bad byte counts		6660	1415	2470	3691
9	HTTP response codes	200	12120	3595	4015	1744
10		301	714	191	249	105
11		302	819	171	273	114
12		403	3843	-	1325	-
13		404	10	-	-	-
14		500	148	-	70	-
15		503	-	341	-	-

Table 1 Performance test result (Server)

Based on the above result, Shuultuur has some advantages and disadvantages. For example, since Shuultuur is still under development, it responded with Internal Server Error (500) more often than Dansguardian. On the other hand, Shuultuur responded with many more successful responses (200). Dansguardian has some limitations, and it responded 341 times with Service Unavailable (503) and had many more timeouts. On the performance side, on average, Shuultuur's performance was higher than Dansguardian in most cases for both tests.

Case 2

The scenario is almost the same as in Case 1, but we used different hardware (APU system board) (20), updated Go to 1.4.1 and changed the Internet link speed to 2Mbps.

The hardware's technical specifications are listed below:

- CPU –AMD G series T40E, 1 GHz dual Bobcat core with 64 bit support, 32K data + 32K instruction + 512K L2 cache per core
- Memory - 4096MB

On APU, we used FreeBSD 10.1-RELEASE and /etc/sysctl.conf includes following:

```
kern.ipc.somaxconn = 4096
kern.maxfiles = 10000
kern.maxfilesperproc = 8500
kern.ipc.maxsockets = 6500
kern.ipc.nmbclusters = 20000
net.inet.tcp.maxtcptw = 4000
```

Because of the smaller hardware we had to change tcp-backlog setting to 4096 in the Redis config file. In this case, we also used HTTP load test using http_load-03feb2015 (parallel and rate test) (18) for both Dansguardian and Shuultuur.

Table 2 shows the comparison of http_load parallel and rate test results.

NO	RESULT NAMES		PARALLEL TEST		RATE TEST	
			SHUULTUUR	DANSGUARDIAN	SHUULTUUR	DANSGUARDIAN
1	Fetches		4319	2643	5877	5225
2	Max parallel		10	10	392	584
3	Mean bytes/ connection		120364	134945	103568	11322.7
4	Fetches/sec		7.19813	4.405	9.795	8.70832
5	Msecs/connect		19.193 mean, 3009.89 max, 0.925 min	6.23727 mean, 53.385 max, 0.991 min	13.3234 mean, 295.472 max, 0.721 min	12.1561 mean, 3023.61 max, 0.903 min
6	Msecs/ first-response		764.861 mean, 59830.3 max, 36.664 min	1337.36 mean, 55849.5 max, 16.704 min	8371.04 mean, 59971.6 max, 36.453 min	35975.6 mean, 59984 max, 56.747 min
7	Timeouts		28	35	329	4618
8	Bad byte counts		1787	2160	3023	4255
9	HTTP response codes	200	3677	2397	4181	542
10		301	9	191	609	-
11		302	366	217	458	70
12		403	233	-	279	-
13		404	-	-	-	-
14		500	5	-	38	-
15		503	-	-	-	-

Table 2 Performance test result (APU)

The test results are similar to what we observed during the tests that were done on the server. As in previous tests, Shuultuur's performance was higher than Dansguardian's in most cases for both tests. Figures 3 and 4 show memory usage on the http_load test on Shuultuur during rate and parallel tests respectively.

Fig. 3: Rate test

Fig. 4: Parallel test

During the test time we captured top reports for both Shuultuur and Dansguardian, which are shown below.

```
SHUULTUUR:
lastpid:  1317;  load averages:  1.52,  1.00,  0.58
71 processes:  1 running, 64 sleeping, 6 stopped
CPU: 31.4% user,  0.0% nice,  5.9% system,  1.6% interrupt, 61.2% idle
Mem: 58M Active, 189M Inact, 158M Wired, 70M Buf, 3519M Free
Swap: 978M Total, 978M Free

PID    USERNAME  THR  PRI  NICE  SIZE    RES     STATE    C  TIME    WCPU     COMMAND
1300   user      18   25   0     84540K  43672K  uwait    1  6:16    91.85%   shuultuur
1299   user      5    21   0     28544K  9484K   piperd   1  0:18    4.10%    gcvis
822    redis     3    52   0     28108K  6540K   uwait    1  0:21    0.29%    redis-server
1024   root      1    20   0     43580K  17092K  select   0  3:42    0.00%    dansguardian
794    squid     1    20   0     164M    68400K  kqread   1  1:20    0.00%    squid
1030   nobody    1    20   0     43580K  18660K  select   1  0:02    0.00%    dansguardian
1028   nobody    1    20   0     43580K  18664K  select   1  0:02    0.00%    dansguardian
1029   nobody    1    20   0     43580K  18672K  select   1  0:02    0.00%    dansguardian
1033   nobody    1    20   0     43580K  18664K  select   0  0:02    0.00%    dansguardian
1032   nobody    1    20   0     43580K  18660K  select   0  0:02    0.00%    dansguardian
1031   nobody    1    20   0     43580K  18672K  select   1  0:02    0.00%    dansguardian
1025   nobody    1    20   0     31292K  5328K   select   1  0:02    0.00%    dansguardian

DANSGUARDIAN:
lastpid:  1151;  load averages:  0.42,  0.68,  0.81
up 0+01:00:20  22:20:41
156 processes: 1 running, 152 sleeping, 3 stopped
CPU:  0.2% user,  0.0% nice, 10.2% system,  1.8% interrupt, 87.8% idle
Mem: 103M Active, 245M Inact, 161M Wired, 58M Buf, 3415M Free
Swap: 978M Total, 978M Free

PID    USERNAME  THR  PRI  NICE  SIZE    RES     STATE    C  TIME    WCPU     COMMAND
1024   root      1    35   0     43580K  17092K  nanslp   0  1:13    23.49%   dansguardian
794    squid     1    26   0     160M    62060K  kqread   0  0:13    4.59%    squid
1002   user      19   42   0     93636K  51320K  STOP     0  9:58    0.00%    shuultuur
1001   user      6    20   0     33856K  10692K  STOP     0  0:32    0.00%    gcvis
822    redis     3    52   0     28108K  6452K   uwait    1  0:15    0.00%    redis-server
932    user      1    20   0     21916K  3244K   CPU0     0  0:06    0.00%    top
1028   nobody    1    20   0     43580K  18152K  select   0  0:01    0.00%    dansguardian
1033   nobody    1    20   0     43580K  18172K  select   0  0:01    0.00%    dansguardian
926    user      1    20   0     86472K  7240K   select   1  0:01    0.00%    sshd
1025   nobody    1    20   0     31292K  5328K   select   1  0:00    0.00%    dansguardian
1030   nobody    1    20   0     43580K  18304K  select   0  0:00    0.00%    dansguardian
1053   nobody    1    20   0     43580K  18664K  select   0  0:00    0.00%    dansguardian
```

As you can see, the system load average, especially CPU usage, was high when Shuultuur was working.

CONCLUSIONS AND FUTURE WORK

Developing an application in Go, using its built-in data structures such as maps and slices, was simple and mostly straightforward. We were able to make the first working prototype in a matter of days. There were many open-source projects written in Go in online source code repositories such as GitHub, and many of those projects were helpful for our development.

The test results are for only two cases. So far, we made http load tests multiple times and results were consistent. We expect that when we reach a first stable version, the results will be a lot better.

As mentioned before, our implementation lacks fast and stable image-checking features. In future work, we will improve image checking and we will have to solve a high number of goroutines problems. Finally, the memory usage and CPU load problem is a major issue for embedded system applications and we are planning to do more research on this to stabilize the resource usages.

BIBLIOGRAPHY

1. ***Half a decade with Go***. Retrieved from The Go blog: http://blog.golang.org/5years
2. Retrieved from Go language resources: http://go-lang.cat-v.org/pure-go-libs
3. ***Go (programming language)***. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Go_%28programming_language%29
4. ***The Go programming language***. Retrieved from http://golang.org/
5. ***Computer Language Benchmarks Game***. Retrieved from http://benchmarksgame.alioth.debian.org/
6. ***String Matching Algorithms and Their Applicability in Various Applications***. (D. G. Nimisha Singla, Ed.) ***International Journal of Soft Computing and Engineering (IJSCE)***, 1 (6).
7. ***Fast Cache for Your Text: Accelerating Exact Pattern Matching with Feed-Forward Bloom Filters.*** School of Computer Science. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University.
8. ***FreeBSD***. Retrieved from https://www.freebsd.org/internet.html
9. ***goproxy.*** (E. Leibovich, Producer) Retrieved from https://github.com/elazarl/goproxy
10. ***gcvis.*** (D. Cheney) Retrieved from https://github.com/davecheney/gcvis
11. ***profile.*** (D. Cheney) Retrieved from https://github.com/davecheney/profile
12. ***go-nude.*** (Koyachi) Retrieved from https://github.com/koyachi/go-nude
13. ***xxxhash-go.*** (S. Bunel) Retrieved from https://bitbucket.org/StephaneBunel/xxhash-go
14. ***powerwalk.*** (Stretchr) Retrieved from https://github.com/stretchr/powerwalk
15. ***redigo.*** (G. Burd) Retrieved from https://github.com/garyburd/redigo
16. ***Redis.*** Retrieved from http://redis.io
17. ***HTTP ListenAndServe Goroutines throughput***. Retrieved from http://grokbase.com/t/gg/golang-nuts/147b9nb2nq/go-nuts-http-listenandserve-goroutines-throughput
18. ***HTTP Load.*** (ACME Lab) Retrieved from http://acme.com/software/http_load/
19. ***Profiling Go programs.*** Retrieved from https://blog.golang.org/profiling-go-programs
20. ***PC engine APU board***. http://www.pcengines.ch/apu1d4.htm
21. ***Goroutines.*** Retrieved from https://golang.org/doc/effective_go.html#goroutines
22. ***Channels.*** Retrieved from https://golang.org/doc/effective_go.html#channels
23. ***Go maps in action.*** Retrieved from https://blog.golang.org/go-maps-in-action
24. ***Arrays, slices (and strings): The mechanics of 'append'***. Retrieved from https://blog.golang.org/slices

Ganbold Tsagaankhuu is a freelancer working on various FreeBSD-related projects. He is also promoting Unix-like operating systems and open source in Mongolia. He is one of the founders of Mongolian Unix User Group. He received his Master degree from Novosibirsk State Technical University (Russia) in 1994. He translated the FreeBSD handbook into Mongolian and started contributing to the FreeBSD project in 2007. From April 2009 to July 2014, he worked for a local mobile operator where he was in charge of an IT division that developed software, administered servers, and improved the security of the company.

Esbold Unurkhaan is a lecturer of Network and System security at the School of Information and Communication Technology (SICT) of the Mongolian University of Science and Technology (MUST), Ulaanbaatar. He received his PhD from Duisburg-Essen University (Germany) in 2005. From 2001 to 2004 he was a researcher at the Institute for Experimental Mathematics of Duisburg-Essen University, where he worked on his PhD thesis, "Secure End-to-End Transport over SCTP—A new security extension for SCTP." After research work in Germany, he returned to Mongolia and worked in the Computer Science and Management School of MUST.

Erdenebat Gantumur is a security engineer at ESCRYPT Inc. He earned his Master of Science in Information Technology and Information Security from Carnegie Mellon University in 2010. He has over 10 years of experience in security areas including network security, information assurance, computer security, and embedded data security. Since 2011, he led and worked on a number of V2X, in-vehicle and smart grid security projects at ESCRYPT Inc. In the past, he has worked as a network engineer, system and network administrator, and information security administrator. He also cofounded the first Mongolian cyber incident response team and has worked as a network security analyst.

How to Install Bacula on FreeBSD

and Get Your First Backup Running

by Dan Langille

ABOUT THIS ARTICLE

In this article you will learn:

- **How to install Bacula**
- **How to get your first backup running**

I WILL BE USING

- **Bacula 7.0.5**
- **PostgreSQL 9.3.5**
- **FreeBSD 9.3**
- **ZFS**

You don't have to be using the same choices to get something out of this article. You will learn Bacula concepts that will help you through commonly misunderstood issues. I promise you'll get something out of it.

Unless otherwise mentioned, all configuration files for Bacula will be in `/usr/local/etc/bacula/`.

Bacula directives ignore spaces. Thus, **WorkingDirectory** and **Working Directory** are identical. Use whichever you prefer.

Some values are in "quotes"; some are not. Use whichever you prefer, but remember that if the value contains a space, "you need to put it in quotes."

Words that are Bacula components or concepts will be in **Bold** with the first letter in **Capital Letters**. This will help you differentiate a **File** directive from a file on disk, or a **Volume** from a volume.

Programs or commands will appear like `bconsole`. Files will appear like `/etc/rc.conf`.

WHAT I WILL COVER

- **Clients**
- **Director**
- **Storage**
- **Backups**
- **Restores**

Full disclosure: I'm biased. I like the combination of PostgreSQL, FreeBSD, ZFS, and Bacula. In fact, I like it so much, I wrote the PostgreSQL backend for Bacula. Since then, I have been using Bacula for over 10 years. I have backed up to HDD, DDS, DLT, and SDLT. Bacula has proven to be reliable and it just works. Clients are available for a wide range of operating systems and Bacula doesn't care what file system you are using.

I am also the FreeBSD port maintainer for Bacula.

I first wrote about Bacula for O'Reilly. I later reposted that article to my diary. I am updating that article for this issue of *FreeBSD Journal*.

There actually isn't much useful information in this post about ZFS, but after writing this article, I found there was so much to cover, and omitting ZFS in details made sense.

Short Introduction of Bacula

This description is short and generalized. Just because I

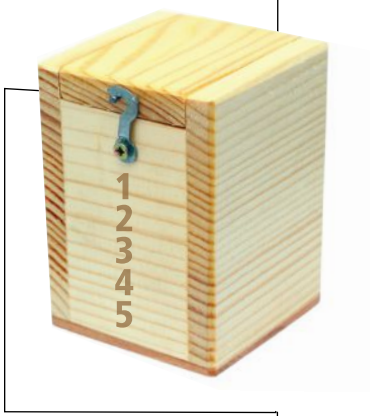
don't mention something, doesn't mean Bacula can't do it.

Bacula is a backup solution based on a networked client-server model. It can back up to disk or tape. The community is supportive and thriving. It has great commercial support if you want that. Control of backups is centralized around a **Director**. When scaling out, you can have as many **Directors** as you want. Each **Director** has **Clients**, which might be shared with other **Directors**.

The **Director** contacts the **Client**, and tells it: back up this FileSet to this **Storage.** The **Client** then does as it is asked and backs up the files. Later, that backup job might be copied or migrated to another **Storage**, which might be local or remote. For example, you might be backing up to disk and you want that job copied to tape. Or copied to a remote location.

Whatever you do with your backups, the **Catalog** keeps track of what was backed up, when, from what **Client**, and where it is located now. The **Catalog** is also used for retention purposes and for pruning and purging.

The Main Components



Bacula is not one program; it is several. The networked programs talk to each other via TCP/IP. The main components of Bacula are:

1. **Director** – daemon (`bacula-dir`) which controls jobs.
2. `bconsole` – command line interface console.
3. **Storage** – daemon (`bacula-sd`) which receives backups from **Clients** and stores them to a storage device (e.g., tape, disk).
4. **Client** – daemon (`bacula-fd`) which reads files and sends them to **Storage**.
5. **Catalog** – a database that keeps track of what jobs have been run and where they are stored. I recommend PostgreSQL for this. MySQL can also be used, but you'll get better performance from PostgreSQL. I do not recommend using SQLite for this purpose.

The rest of this section can be skipped. It is not vital to understanding how to get Bacula running for your first time. All of these components need to be able to talk to each other. Keep this in mind with respect to your firewalls. Especially when you read the following and see the order of the communications.

- `bacula-dir` contacts `bacula-fd` to tell it what to back up
- `bacula-fd` contacts `bacula-sd` to send it backups
- `bacula-sd` contacts `bacula-dir` with the status of completed backups
- `bacula-dir` contacts the database server with the status
- `bconsole` contacts `bacula-dir`
- You can query status of `bacula-dir`, `bacula-sd`, and `bacula-fd` via `bconsole`. When you do this, `bacula-dir` contacts `bacula-sd`, and then `bacula-fd` to ask their status and relay it back to you
- If copying jobs from one storage to another, the `bacula-sd` needs to talk to the other `bacula-sd`
- If you use the **SD Calls Client** directive, `bacula-sd` needs to be able to contact `bacula-fd`

My Backups

When I first started with Bacula, I backed up directly to tape. Now, with HDD being larger and cheaper, I back up to disk. Some people think that's enough. I don't. One power supply failure could wipe all those HDD. ZFS is the best file system there is, but it does not protect against catastrophic tragedies. I believe in multiple copies of your backups. At least one here, and one there. This is why I back up to tape, and then move those tapes to another location.

I have plans to send backups to a remote `bacula-sd` as well. I have not implemented that yet, but I will. Soon.

Installation

Installing Bacula is rather easy. If you want to use PostgreSQL, and I recommend you do, you can install from the pre-built packages. If not, install from the ports tree.

Here is what I installed:

```
# pkg install bacula-server
# pkg install postgresql93-server
```

I am using PostgreSQL 9.3 as that is the current default in the FreeBSD ports/packages system. Other versions will be fine.

Database Setup

This is a lightweight introduction to configuring PostgreSQL for use with Bacula. The official documentation is much more complete.

In these steps, we first create a new PostgreSQL database cluster and then start the server:

```
# service postgresql initdb

The files belonging to this database system will be owned by user "pgsql".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /usr/local/pgsql/data ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
creating configuration files ... ok
creating template1 database in /usr/local/pgsql/data/base/1 ... ok
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects' descriptions ... ok
creating collations ... ok
creating conversions ... ok
creating dictionaries ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
loading PL/pgSQL server-side language ... ok
vacuuming database template1 ... ok
copying template1 to template0 ... ok
copying template1 to postgres ... ok
syncing data to disk ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    /usr/local/bin/postgres -D /usr/local/pgsql/data
or
    /usr/local/bin/pg_ctl -D /usr/local/pgsql/data -l logfile start

# service postgresql start
LOG:  could not create IPv6 socket: Protocol not supported
LOG:  ending log output to stderr
HINT:  Future log output will go to log destination "syslog".
```

Next, we will create the database, the **Catalog**, for Bacula to use. Some output has been trimmed from this example because it is repetitive and not very exciting to read.

```
# su pgsql
# cd cd/usr/local/share/bacula/
$ createuser -d bacula
$./create_bacula_database
Creating postgresql database
CREATE DATABASE
ALTER DATABASE
Creation of bacula database succeeded.
Database encoding OK
$ psql -l
                    List of databases
Name       | Owner    |Encoding    |Collate   | Ctype  | Access privileges
-----------+---------+-------------+---------+-------+-------------------
bacula     | pgsql    |SQL_ASCII   |C         | C      |
postgres   | pgsql    |UTF8        |C         | C      |
template0  | pgsql    |UTF8        |C         | C      | =c/pgsql+pgsql=CTc/pgsql
template1  | pgsql    |UTF8        |C         | C      | =c/pgsql+pgsql=CTc/pgsql
(4 rows)


$./make_bacula_tables
Making postgresql tables
CREATE TABLE
ALTER TABLE
CREATE INDEX
CREATE TABLE
ALTER TABLE

[some output trimmed]

INSERT 0 1
INSERT 0 1
INSERT 0 1

[some output trimmed]

INSERT 0 1
Creation of Bacula PostgreSQL tables succeeded.
$ ./grant_bacula_privileges
Granting postgresql privileges
psql::2: ERROR:  role "bacula" already exists
GRANT
GRANT
GRANT

[some output trimmed]

GRANT
GRANT
GRANT
Privileges for user bacula granted on database bacula.
```

This is what you should see now:

```
$ psql -l
                                 List of databases
Name        | Owner   | Encoding  | Collate  | Ctype   |Access privileges
------------+---------+-----------+----------+---------+-------------------
bacula      | pgsql   | SQL_ASCII | C        | C       |
postgres    | pgsql   | UTF8      | C        | C       |
template0   | pgsql   | UTF8      | C        | C       |=c/pgsql   +pgsql=CTc/pgsql
template1   | pgsql   | UTF8      | C        | C       |=c/pgsql   +pgsql=CTc/pgsql
(4 rows)
```

Configuration

All of this configuration will be done on a single IP address. For starting out, I recommend this approach. This will let you get things running first, then move them to another server. One step at a time.

The IP address I am using is 10.55.0.76 and that is the only value you should have to change in your configuration if everything is installed on one system.

The are four main configuration files with Bacula. The default location is `/usr/local/etc/bacula/`:

1. `bacula-dir.conf`
2. `bacula-fd.conf`
3. `bacula-sd.conf`
4. `bconsole.conf`

These file names are self-explanatory. In the next few sections, I will give you a minimal configuration which will get you started. I guarantee it.

In the following examples, we will start slowly and quickly gain speed. The first `bacula-dir.conf` file you see will be added to as we go along. I will provide links for all configuration files at the end of this post.

bacula-dir.conf

`bacula-dir.conf` is, by far, the largest configuration file you will create for Bacula. Fortunately, you can break it up into logical areas and keep them in as many, or as few, files as you like. Amongst other things, the items specified by `bacula-dir.conf` include:

- identity of this particular `bacula-dir`
- identify the **Client** it will back up
- list the **Job** which will run on that **Client**
- specify the **Catalog** which will document those backups
- list the **FileSet** (list of files) for that **Job**
- specify the **Schedule** for that **Job**
- identify the **Storage** to be used

There are other directives (e.g., such as Pools) but we will postpone that discussion until later.

I like to align my configuration files like this. I find it easier to read. It is not required.

```
1   Director {
2     Name                  = MyBaculaDirector
3     Password              = "the bacula-dir password"
4
5     QueryFile             = "/usr/local/share/bacula/query.sql"
6
7     PidDirectory          = "/var/run"
8     WorkingDirectory      = "/usr/local/bacula/working"
9
10    Messages              = MyMessages
11    }
12
13    Catalog {
14    Name                  = MyCatalog
15    dbname                = bacula
```

CONTINUES NEXT PAGE

CONTINUED

```
16   dbaddress          = 10.55.0.76
17   user               = bacula
18   password           = ""
19   }
20
21   Messages {
22   Name               = MyMessages
23   mailcommand        = "/usr/local/sbin/bsmtp -h localhost -f
                          \"\(Bacula\)%r\" -s\"Bacula: %t %e of %c %l\" %r"
24   operatorcommand    = "/usr/local/sbin/bsmtp -h localhost -f \"\(Bacula\)%r\"
                          -s \"Bacula: Intervention needed for %j\" %r"
25
26   operator           = root@localhost = mount
27   mail               = root@localhost = all
28   console            = all
29   append             = "/var/log/bacula.log" = all, !skipped, !restored
30   catalog            = all, !skipped, !saved
31   }
```

You can use this configuration as shown. I recommend keeping that password for your first setup. Change it later, after you have everything working.

That **Name** and **Password** are the credentials which this **Director** will accept when you login via `bconsole`. Those values will appear in two places:

1. in `bacula-dir.conf`, so `bacula-dir` knows what credentials to accept
2. in `bconsole.conf`, so `bconsole` knows what credentials to send

This "appear in two places" is an important concept to learn early on. It is a design used by Bacula in several places. I will point these out when they appear in later examples.

As your `bacula-dir.conf` file grows, you can split out your `bacula-dir.conf` configuration into separate files. Each can be included with a command such as this:

```
1  @/usr/local/etc/bacula/client-myclient.conf
```

I will make use of this feature later on.

Lines 13-19 identify the database to be used for the **Catalog**. It does not specify what type of database server (e.g., PostgreSQL); that option is set at compile time.

Lines 21-31 specify how messages should get to you. This configuration assumes you have an SMTP server on your system, but you will not need it to follow this example. It is recommended if you start to depend upon your backups.

bacula-fd.conf

Here is a minimal `bacula-fd.conf`:

```
 1  Director {
 2    Name              =   MyBaculaDirector
 3    Password          =   "the bacula-fd password"
 4    }
 5
 6    FileDaemon {
 7    Name              =   MyFirstBaculaClient
 8    WorkingDirectory  =   /var/db/bacula
 9    Pid Directory     =   /var/run
10    }
11
12    Messages {
13    Name              =   Standard
14    director          =   MyBaculaDirector = all, !skipped, !restored
15  }
```

As with `bacula-dir.conf`, the **Name** and **Password** represent credentials. However, in this case, the **Name** and **Password** identify one **Director** which can contact this **Director**. Combined, these two values are the credentials required to communicate with this `bacula-fd`. It will be stored in two places:

1. this `bacula-fd.conf`, so this `bacula-fd` knows what credentials to accept
2. in `bacula-dir.conf`, so the `bacula-dir` knows what credentials to present when contacting this `bacula-fd.`

I will show item #2 later.

An advanced setup can have multiple **Directors** in which case, a given `bacula-fd.conf` can have multiple **Director** directives, but we won't go there in this simple example.

bacula-sd.conf

This is our minimal `bacula-sd.conf`:

```
1  Storage {
2    Name             =  MyBaculaStorage
3    WorkingDirectory =  "/usr/local/bacula/working"
4    Pid Directory    =  "/var/run"
5    }
6
7    Director {
8    Name             =  MyBaculaDirector
9    Password         =  "the bacula-sd password"
10   }
11
12   Device {
13   Name             =  MyFirstStorageDevice
14   Media Type       =  MyMediaType
15   Archive Device   =  /usr/local/bacula/volumes
16   LabelMedia       =  yes
17   Random Access    =  yes
18   AutomaticMount   =  yes
19   RemovableMedia   =  no
20   AlwaysOpen       =  no
21   }
22
23   Messages {
24   Name             =  Standard
25   director         =  MyBaculaDirector = all
26   }
```

I think you see the pattern now. That **Name** and **Password** identifies a **Director** which can contact us. That set of credentials will be stored:

1. in `bacula-sd.conf` so `bacula-sd` knows what credentials must be presented,
2. in `bacula-dir.conf` so `bacula-dir` knows what credentials to present to this `bacula-sd.`

As with `bacula-fd.conf`, we'll get back to item #2 later.

bconsole.conf

This minimal `bconsole.conf` will let you interact with `bacula-dir`:

```
1  Director {
2    Name             =  MyBaculaDirector
3    Password         =  "the bacula-dir password"
4
5    Address          =  10.55.0.76
6    }
```

This **Name** and **Password** are the same that appear in `bacula-dir.conf`. These are the previously mentioned credentials required to talk to `bacula-dir`. This file will be read by `bconsole` when it starts up.

That's Just the Start

In the previous sections, you have seen the bare basics. In the following sections, you'll see how it is all tied together. I will show you how the **Director** knows about **Clients**, **Jobs**, and **FileSets**. So far, we have just been defining the foundation. Now we start building our backup system.

Client

In this section, we will add a **Client** to the Bacula configuration. This could be added directly to `bacula-dir.conf`, but I prefer to put everything specifically related to a **Client** in a separate file.

I am putting the following into `client-myclient.conf`:

```
1  Client {
2    Name            = MyFirstBaculaClient
3    Password        = "the bacula-fd password"
4    Address         = 10.55.0.76
5    Catalog         = MyCatalog
6    }
```

To pull this configuration into `bacula-dir.conf`, I added the following to `bacula-dir.conf`:

```
1  @/usr/local/etc/bacula/client-myclient.conf
```

This configuration will tell `bacula-dir` to include the file, and within that file will be the contact details for our `bacula-fd`.

Remember how the credentials are stored in two places? One for the sender, one for the receiver? Here is an example to demonstrate that:

```
1  # grep "the bacula-fd password" *
2  bacula-fd.conf:  Password          =     "the bacula-fd password"
3  client-myclient.conf:  Password    =     "the bacula-fd password"
```

The first instance tells `bacula-fd` what password to accept. The second tells `bacula-dir` what credentials to use when contacting that `bacula-fd`.

Please note: Whenever a **Director** contacts a **Client**, the credentials it must use are the **Director**'s **Name** (not the **Client**'s **Name**) and the **Password** specified in the **Client** resource in `bacula-dir.conf`. To be clear, in our example, the correct values that the **Director** must pass to the **Client** are:

1. `MyBaculaDirector`
2. `the bacula-fd password`

Thus, the **Name** in a **Client** resource names the client and does not play a role in the credentials. This is very important.

FileSet

Now that we have told Bacula about our **Client**, we will specify the files we want to back up: a FileSet.

I put the following into `filesets.conf`.

```
1  FileSet {
2  Name                =  "MyFirstFileSet"
3  Include {
4  Options {
5  signature           =  MD5
6  }
```

CONTINUES NEXT PAGE

CONTINUED

```
7
8   File                    =  /usr/local/etc/bacula/
9   }
10
11  Exclude {
12  File                    =  *~
13  }
14  }
```

In this **FileSet**, we are backing up one directory. Do not be confused by that name, **File**. I've chosen to back up `/usr/local/etc/bacula/` because I know you'll have that.

Each file will have its MD5 recorded. This is optional, but highly recommended. It is used when running a **Verify Job** to compare what you have on disk to what you backed up.

We will be excluding any files ending with a ~ (tilde).

Note:

- one **FileSet** per job,
- a given **FileSet** can be used by more than one **Job.**

To tell `bacula-dir` about this **FileSet**, I added this line to `bacula-dir.conf`:

```
1   @/usr/local/etc/bacula/filesets.conf
```

The next step: create a **Backup Job**.

Job

A **Job** identifies what is to be backed up, the **Client** it is located on, and the `bacula-sd` which will store this backup. This is what I added to `client-myclient.conf`:

```
1   Job {
2   Name                        = "MyFirstJob"
3   Client                      = MyFirstBaculaClient
4   Type                        = "Backup"
5   FileSet                     = "MyFirstFileSet"
6   Storage                     = MyFirstStorage
7   Schedule                    = MyFirstSchedule
8   Messages                    = MyMessages
9
10  Pool                        = FullFile # required parameter for all Jobs,
                                  despite what appears in the next few lines
11
12  Full Backup Pool            = FullFile
13  Differential Backup Pool    = DiffFile
14  Incremental Backup Pool     = IncrFile
15  }
```

Now we have told Bacula nearly everything: what to back up, from where, and where to store it. Now we have to tell Bacula when to run this **Job**.

Schedule

A Schedule is very flexible. We will start with simple and classic. The following tells us when the Job will be run, and at what Level (Incremental, Differential, Full). I put this into `schedules.conf`.

```
1   Schedule {
2   Name                    = MyFirstSchedule
3   Run                     = Level=Full 1st sun at 8:15
4   Run                     = Level=Differential 2nd-5th sun at 8:15
5   Run                     = Level=Incremental mon-sat at 8:15
6    }
```

I added this line to **bacula-dir.conf** to include the above:

```
1 @/usr/local/etc/bacula/schedules.conf
```

My server is on UTC time, so these will run at about 3:15 AM EST (I live just outside Philadelphia).

1. On the first Sunday of every month, this will run a Full backup.
2. On all other Sundays, this will run a Differential backup.
3. Every day, we will run an Incremental backup.

The best description for Level is in the Job Resource documentation.

Storage

In that **Job**, we told Bacula where to send the backups: *MyFirstStorage*. We have already defined this **Storage** in **bacula-sd.conf**, but we have not told **bacula-dir.conf** about it. This may sound odd. Why do we have to specify it twice? Bacula is built upon components. In this example, everything is running on the same computer. However, they can easily run on different computers. Therefore, the **bacula-dir.conf** must identify the **bacula-sd** it can use. **bacula-sd.conf** might be on another computer.

Here is how I told **bacula-dir** about the **bacula-sd** it should use. I added this to **bacula-dir.conf**:

```
1 Storage {
2 Name        = MyFirstStorage
3 Address     = 10.55.0.76
4 Password    = "the bacula-sd password"
5 Device      = MyFirstStorageDevice
6 Media Type  = MyMediaType
7 }
```

Here is that concept of **Name** and **Password** again. But this time, it's something slightly different. In this case, we are specifying the credentials **bacula-dir** should use when contacting a **bacula-sd** at that **Address**.

The values for **Device** and **Media Type** must match those specified in **bacula-sd.conf.**

I want to repeat something I mentioned when I added the **Client** resource to **bacula-dir.conf**. The **Name** of this **Storage** resource does not form part of any credentials. When our **Director** contains the **bacula-sd** at 10.55.0.73, it will use these credentials:

1. **MyBaculaDirector**
2. **the bacula-sd password**

Pools & Volumes

So far, I have glossed over Pools. It is easier to introduce **Pools** by describing tapes. Imagine you have a box of tapes for **Full** backups. Every tape in that box will be kept for 12 months. You have another box for **Incremental** backups. Each of those will be kept for 15 days. Another box has **Differential** backups, each of which is kept for 3 months.

Each of those boxes is a **Pool**. Each of those tapes is a **Volume**.

Do not confuse a Bacula **Volume** with a file system or another concept you may have of a volume. A Bacula **Volume** contains a backup (or part of a backup), or it might be empty.

A **Pool** definition is used to describe the initial and/or common characteristics of a **Volume**. Thus, a **Pool** can be thought of as a collection of **Volumes** which share common characteristics, such as **Retention**.

These are the **Pool** definitions I added to **pools.conf**:

```
1 Pool {
2 Name         =  FullFile
3 Pool Type    =  Backup
```

CONTINUES NEXT PAGE

CONTINUED

```
Recycle                 =   yes
AutoPrune               =   yes
Volume Retention        =   3years
Storage                 =   MyFirstStorage

Maximum Volume Bytes    =   5G
Maximum Volumes         =   5

LabelFormat             =   "FullAuto-"
}

Pool {
Name                    =   DiffFile
Pool Type               =   Backup
Recycle                 =   yes
AutoPrune               =   yes
Volume Retention        =   6 weeks
Storage                 =   MyFirstStorage

Maximum Volume Bytes    =   5G
Maximum Volumes         =   5

LabelFormat             =   "DiffAuto-"
}

Pool {
Name                    =   IncrFile
Pool Type               =   Backup
Recycle                 =   yes
AutoPrune               =   yes
Volume Retention        =   3 weeks
Storage                 =   MyFirstStorage

Maximum Volume Bytes    =   5G
Maximum Volumes         =   5

LabelFormat             =   "IncrAuto-"
}
```

This specifies three pools, one for each **Level** of backup, and each with different values for **Retention**.

If you want to skip ahead to running jobs, you can ignore the next few sections.

Pool limits

You will notice that each **Pool** I created has two maximum settings. Bacula will keep using a **Volume** forever unless you tell it otherwise. I have chosen to restrict my **Pools** by specifying **Maximum Volume Bytes**. When a **Volume** reaches this size, Bacula will stop using it and create another. This automatic creation can occur because of these directives:

1. the **Pool** has a **LabelFormat,**
2. the **Device** in `bacula-sd.conf` has **LabelMedia** set to **yes.**

I have also specified a maximum of 5 **Volumes** in each **Pool**. This limits the size of each **Pool** to 25G. Clearly, these are just values for getting started. Set your values accordingly once you get things running.

There are many such options in the Pool documentation.

Retention

When you first think of Retention, most people consider how long the backup is kept. With respect to Bacula, this is not strictly the correct definition. Bacula **Retention** refers to the **Catalog** and

specifies how long meta-data will be retained in the database. The **Catalog** records what **File** was backed up by what **Job** and the **Volume** it is located upon.

Bacula uses three **Retention** periods:

1. File Retention
2. Job Retention
3. Volume Retention

File Retention and **Job Retention** are specified in each **Client** resource of `bacula-dir.conf`. I have not specified any for my **Client**, and they therefore default to 60 and 180 days, respectively.

Volume Retention is specified in the **Pool** definition. Of vital importance is the starting point for **Volume Retention**. The **Volume Retention** period starts when a **Volume** is no longer appendable. The countdown for **Volume Retention** begins only when the **Volume Status** has been changed to something other than **Append** (e.g., **Full**, **Used**, **Purged**, …).

Meta-data is removed from the **Catalog** by Pruning. You will remember that each of my **Pools** has **AutoPrune = yes**.

Keep aware that once **File Retention** has passed for a particular backup (and the **Catalog** has been pruned), you will be unable to select the files you wish to restore from that backup. You will need to restore the entire **Job**, and once **Job Retention** (for that backup) has passed (and the **Catalog** has been pruned), you will be unable to restore that backup without restoring to `bextract` or `bscan`. If you have to do that, it will not be the highlight of your week.

Disk space is cheap. Time is not. Keep your **File** and **Job Retention** set high to make your restore easier. I think my **Catalog** database is about 10GB when dumped and compressed. I recommend not trying to skimp on disk space by reducing **Job** or **File Retention** unnecessarily.

I set **File** and **Job Retention** to large values. Then I let **Volume Retention** determine how long the metadata is retained. Thus, I can copy the same backup to different **Pools** and let that **Pool Volume Retention** decide how long that backup will be retained.

Restoring Pruned Data

If **Pruning** has occurred, you can use `bscan`. I hope you never have to use it under emergency conditions. I view `bscan` as a tool of last resort when everything else has been lost.

Recycling

By design, Bacula avoids overwriting backups. It will do everything it can to avoid this. You must impose restrictions which indicate that Bacula is allowed to overwrite. If you do not do this, Bacula will continue appending to existing **Volumes** until it runs out of space. NOTE: disk space monitoring is outside the scope of this post; use Nagios.

Once you impose restrictions, Recycling can occur and Bacula can then overwrite a **Volume**. The details of those restrictions are defined in the documentation. I will not go into **Recycling** here, but the examples given here are very restrictive.

Running a Job

Let's run our first job. Oh… Umm, we have some last minute things to do.

Last Minute Things

The following directory is used by both `bacula-dir` and `bacula-sd`:

```
mkdir -p /usr/local/bacula/working
chown bacula:bacula /usr/local/bacula/working
```

The following log is used by `bacula-dir`:

```
touch /var/log/bacula.log
chown bacula:bacula /var/log/bacula.log
```

The following allows incoming database connections. You may wish to tighten this down later, and

perhaps use a password. I added this entry to the end of `/usr/local/pgsql/data/pg_hba.conf`:

```
host    bacula          bacula          10.55.0.76/32            trust
```

This is where `bacula-sd` is configured to store the backups. This is where your **Volumes** will be stored. This is set in `bacula-sd.conf`.

```
mkdir -p /usr/local/bacula/volumes
chown bacula:bacula /usr/local/bacula/volumes
```

Starting the Daemons

With these commands, I started the Bacula daemons:

```
# service bacula-fd start
# service bacula-sd start
# service bacula-dir start
```

They can be started in any order.

Running bconsole

`bconsole` is your friend. You can run backups and restores with `bconsole`. Here is what you should see when you start it:

```
# bconsole
Connecting to Director 10.55.0.76:9101
1000 OK: 1 MyBaculaDirector Version: 7.0.5 (28 July 2014)
Enter a period to cancel a command.
*
```

If you don't see that, you might want to verify that `bacula-dir` is running.

Status

In this section, we will run status on the **Director**, **Client**, and **Storage**.

```
*status director
MyBaculaDirector Version: 7.0.5 (28 July 2014) amd64-portbld-freebsd9.1
 freebsd 9.1-RELEASE-p19
Daemon started 10-Jan-15 20:14. Jobs: run=0, running=0 mode=0,0
Heap: heap=0 smbytes=253,680 max_bytes=254,453 bufs=168 max_bufs=176

Scheduled Jobs:
Level            Type     Pri      Scheduled     Job Name    Volume
===================================================================================
Differential     Backup   10       11-Jan-15     08:15       MyFirstJob  *unknown*
====

Running Jobs:
Console connected at 10-Jan-15 20:20
No Jobs running.
====
No Terminated Jobs.
====
 *
```

You can abbreviate commands. I could have entered `st di` and received the same output.

```
*st client
Automatically selected Client: MyFirstBaculaClient
```

CONTINUES NEXT PAGE

CONTINUED

```
Connecting to Client MyFirstBaculaClient at 10.55.0.76:9102

MyFirstBaculaClient Version: 7.0.5 (28 July 2014)     amd64-portbld-freebsd9.1 freebsd 9.1-RELEASE-p19
Daemon started 10-Jan-15 20:14. Jobs: run=0 running=0.
Heap: heap=0 smbytes=185,618 max_bytes=185,765 bufs=50 max_bufs=51
Sizes: boffset_t=8 size_t=8 debug=0 trace=0 mode=0,0 bwlimit=0kB/s

Running Jobs:
Director connected at: 10-Jan-15 20:22
No Jobs running.
====

Terminated Jobs:
====
*
```

Because we have only one **Client**, `bacula-dir` selected that **Client** automatically and showed us the status. When executing the command, `bacula-dir` connects to `bacula-fd` and asks it for the status. Seeing this type of output confirms that `bacula-dir` had the correct credentials for that client. This situation is vital to running a job.

Let's check the storage.

```
*st stor
Automatically selected Storage: MyFirstStorage
Connecting to Storage daemon MyFirstStorage at 10.55.0.76:9103

MyBaculaStorage Version: 7.0.5 (28 July 2014) amd64-portbld-freebsd9.1
freebsd 9.1-RELEASE-p19
Daemon started 10-Jan-15 20:26. Jobs: run=0, running=0.
Heap: heap=0 smbytes=187,799 max_bytes=312,674 bufs=56 max_bufs=57
Sizes: boffset_t=8 size_t=8 int32_t=4 int64_t=8 mode=0,0

Running Jobs:
No Jobs running.
====

Jobs waiting to reserve a drive:
====

Terminated Jobs:
====

Device status:

Device "MyFirstStorage" (/usr/local/bacula/volumes) is not open.
==
====

Used Volume status:
====

====

*
```

Do not be concerned with the `'Device "MyFirstStorage" (/usr/local/bacula/volumes) is not open'` message. This is completely normal when a job is not underway.

Your First Job

Let's run a **Job**.

```
*run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
Automatically selected Job: MyFirstJob
Run Backup job
JobName:          MyFirstJob
Level:            Incremental
Client:           MyFirstBaculaClient
 FileSet:         MyFirstFileSet
Pool:             FullFile (From Job resource)
Storage:          MyFirstStorage (From Pool resource)
When:             2015-01-10 20:35:08
Priority:         10
OK to run?        (yes/mod/no): yes
Job queued. JobId=4
You have messages.
*m
10-Jan 20:35      MyBaculaDirector JobId 4: No prior Full backup Job record found.
10-Jan 20:35      MyBaculaDirector JobId 4: No prior or suitable Full backup
                  found in catalog. Doing FULL backup.
*m
10-Jan 20:35      MyBaculaDirector JobId 4: Start Backup JobId 4,Job=MyFirstJob.
                  2015-01-10_20.35.10_03
10-Jan 20:35      MyBaculaDirector JobId 4: Created new Volume="FullAuto-0001",Pool=
                  "FullFile", MediaType="MyMediaType" in catalog.
10-Jan 20:35      MyBaculaDirector JobId 4: Using Device "MyFirstStorageDevice"
                  to write.
10-Jan 20:35      MyBaculaStorage JobId 4: Labeled new Volume "FullAuto-0001" on
                  file device "MyFirstStorageDevice" (/usr/local/bacula/volumes).
10-Jan 20:35      MyBaculaStorage JobId 4: Wrote label to prelabeled Volume "FullAuto-
                 0001" on file device "MyFirstStorageDevice" (/usr/local/bacula/volumes)
*m
You have no messages.
*m
You have no messages.
*m
You have no messages.
*m
10-Jan 20:35        MyBaculaStorage JobId 4: Elapsed time=00:00:10, Transfer
                    rate=4.517 K Bytes/second
*m
10-Jan 20:35        MyBaculaDirector JobId 4: Bacula MyBaculaDirector 7.0.5 (28Jul14):
   Build OS:        amd64-portbld-freebsd9.1 freebsd 9.1-RELEASE-p19
   JobId:           4
   Job:             MyFirstJob.2015-01-10_20.35.10_03
   Backup Level:    Full (upgraded from Incremental)
   Client:          "MyFirstBaculaClient" 7.0.5 (28Jul14) amd64-port
                    bld-freebsd9.1,freebsd,9.1-RELEASE-p19
   FileSet:         "MyFirstFileSet" 2015-01-10 20:27:51
   Pool:            "FullFile" (From Job FullPool override)
   Catalog:         "MyCatalog" (From Client resource)
   Storage:         "MyFirstStorage" (From Pool resource)
   Scheduled time:  10-Jan-2015 20:35:08
   Start time:      10-Jan-2015 20:35:13
   End time:        10-Jan-2015 20:35:23
   Elapsed time:    10 secs
   Priority:        10
   FD Files Written:20
   SD Files Written:20
   FD Bytes Written:42,576 (42.57 KB)
   SD Bytes Written:45,179 (45.17 KB)
```

CONTINUES NEXT PAGE

CONTINUED

```
55      Rate:                       4.3 KB/s
56      Software Compression:       None
57      VSS:                        no
58      Encryption:                 no
59      Accurate:                   no
60      Volume name(s):             FullAuto-0001
61      Volume Session Id:          2
62      Volume Session Time:        1420921921
63      Last Volume Bytes:          46,495 (46.49 KB)
64      Non-fatal FD errors:        0
65      SD Errors:                  0
66      FD termination status:      OK
67      SD termination status:      OK
68      Termination:                Backup OK
69
70  10-Jan 20:35 MyBaculaDirector JobId 4: Begin pruning Jobs older than 6 months .
71  10-Jan 20:35 MyBaculaDirector JobId 4: No Jobs found to prune.
72  10-Jan 20:35 MyBaculaDirector JobId 4: Begin pruning Files.
73  10-Jan 20:35 MyBaculaDirector JobId 4: No Files found to prune.
74  10-Jan 20:35 MyBaculaDirector JobId 4: End auto prune.
75
76  *
```

There. Done. Your first backup!

However, see JobId 4? Yes, it took me four attempts to get the first job running. I had stuff to fix up and adjust in the configuration. You should have no such problems if you copy the configuration files, adjust the IP addresses, and run through all the last minute steps.

Here are some notes to help you understand the output. The numbers refer to the line numbers in the above output.

18 – m is short for messages, and will display any pending messages from the **Director**
20 – The job is promoted to **Full**. This happens when no **Jobs** exist upon which an **Incremental** or **Differential** can be based.
23 – A new **Volume** was created in the **FullFile Pool**, with the specified label format. Because we are backing up to disk, the file name will match the **Volume** name.
25 – The label is also stored inside the **Volume**.
51-52 – 20 files were saved. You might wonder, why 20 files? That's way more than in my configuration. Let's look into that:

```
[root@baculatest:/usr/local/etc/bacula] # find .
.
./INSTALLED
./INSTALLED/bacula-dir.conf
./INSTALLED/bacula-dir.conf.sample
./INSTALLED/bacula-sd.conf.sample
./INSTALLED/bacula-barcodes.sample
./INSTALLED/bacula-fd.conf
./INSTALLED/bacula-sd.conf
./INSTALLED/bacula-fd.conf.sample
./INSTALLED/bacula-barcodes
./INSTALLED/bconsole.conf
./INSTALLED/bconsole.conf.sample
./bacula-dir.conf~
./bacula-dir.conf
./filesets.conf
./pools.conf
./bacula-fd.conf
./client-myclient.conf
./schedules.conf
```

CONTINUES NEXT PAGE

CONTINUED

```
./bacula-sd.conf
./bconsole.conf
```

I moved the files installed by default into a subdirectory, `INSTALLED`. How many files are there?

```
[root@baculatest:/usr/local/etc/bacula] # find . | wc -l
       21
```

21 files? Only 20 backed up? Bacula ignored `bacula-dir.conf~` as specified in the **FileSet**.

Restore

You need only one Restore Job. Restores are done manually, although you could script that. All **Job** parameters can be altered from within `bconsole`. This means you do not have to specify all possible combinations you might need.

```
*restore
Using Catalog "MyCatalog"
No Restore Job Resource found in bacula-dir.conf.
You must create at least one before running this command.
*
```

Oh. Let's add that **Restore Job** to `bacula-dir.conf`.

```
Job {
Name                = "RestoreFiles"
Type                = Restore
Client              = MyFirstBaculaClient
FileSet             = MyFirstFileSet
Storage             = MyFirstStorage
Schedule            = MyFirstSchedule
Messages            = MyMessages
Pool                = FullFile

Where               = /tmp/bacula-restores
}
```

Even though this **Job** refers to *MyFirstBaculaClient*, I consciously did not add this **Job** to `client-myclient.conf` because it can be used for any **Client**.

Notice that the **Restore Job** contains a **Where** directive. This specifies a prefix to the directory name of all files being restored. This permits files to be restored in a different location from which they were backed up. If **Where** is not specified, the files are restored to their original location. Where can also be specified/overridden from within `bconsole` (see the example to follow).

I like to always specify a **Where** directive as a safety/sanity precaution.

Now. Let's run that **Restore**.

```
*restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
1: List last 20 Jobs run
2: List Jobs where a given File is saved
3: Enter list of comma separated JobIds to select
```

CONTINUES NEXT PAGE

CONTINUED

```
 4: Enter SQL list command
 5: Select the most recent backup for a client
 6: Select backup for a client before a specified time
 7: Enter a list of files to restore
 8: Enter a list of files to restore before a specified time
 9: Find the JobIds of the most recent backup for a client
 10: Find the JobIds for a backup for a client before a specified time
 11: Enter a list of directories to restore for found JobIds
 12: Select full restore to a specified Job date
 13: Cancel
 Select item:  (1-13):
```

Of the options listed, I recommend #5 and #6. The others are great, but those are the two I use the most.

We have just one backup, so I will select #5:

```
Select item:  (1-13): 5
Automatically selected Client: MyFirstBaculaClient
Automatically selected FileSet: MyFirstFileSet
+--------+--------+----------+----------+------------+-----------+
|jobid   | level  | jobfiles | jobbytes | starttime  | volumename |
+--------+--------+----------+----------+------------+-----------+-------------+
|4       | F      | 20       | 42,576   | 2015-01-10 20:35:13    | FullAuto-0001|
+--------+--------+----------+----------+------------+-----------+-------------+
You have selected the following JobId: 4

Building directory tree for JobId(s) 4 ...
18 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$
```

Bacula has automatically selected both the **Client** and **FileSet**, since there is only one.

18 files? I thought we backed up 20 files? Yes we did, and two of those "files" were directories. At this point, we are in a special Bacula "shell", where we can move around in the directories and select the files we want to restore. We could have specified **restore all** to automatically mark all files for restore, or we can issue the **restore all** command now.

I will mark all the `*.conf files` in `/usr/local/etc/bacula`

```
$ ls
usr/
$ cd usr/local/etc
cwd is: /usr/local/etc/
$ ls
bacula/
$ cd bacula
cwd is: /usr/local/etc/bacula/
$ ls
INSTALLED/
bacula-dir.conf
bacula-fd.conf
bacula-sd.conf
bconsole.conf
client-myclient.conf
filesets.conf
pools.conf
```

CONTINUES NEXT PAGE

CONTINUED

```
schedules.conf
$ mark *.conf
8 files marked.
$ done
Bootstrap records written to
/usr/local/bacula/working/MyBaculaDirector.restore.2.bsr


The Job will require the following (*=>InChanger):
Volume(s)                 Storage(s)                  SD Device(s)
===========================================================================

FullAuto-0001             MyFirstStorage              25 MyFirstStorageDevice

Volumes marked with "*" are in the Autochanger.


8 files selected to be restored.

Run Restore job
JobName:                  RestoreFiles
Bootstrap:                /usr/local/bacula/working/MyBaculaDirector.restore.2.bsr
Where:                    /tmp/bacula-restores
Replace:                  always
FileSet:                  MyFirstFileSet
Backup Client:            MyFirstBaculaClient
Restore Client:           MyFirstBaculaClient
Storage:                  MyFirstStorage
When:                     2015-01-10 21:17:48
Catalog:                  MyCatalog
Priority:                 10
OK to run? (yes/mod/no):
```

You can now override any **Job** directive with the **mod** command. I will just say **yes** and let the files go to **/tmp/bacula-restores**.

```
OK to run? (yes/mod/no): yes
Job queued. JobId=5
*m
10-Jan 21:20 MyBaculaDirector JobId 5: Start Restore Job RestoreFiles.
2015-15  01-10_21.20.07_04
10-Jan 21:20 MyBaculaDirector JobId 5: Using Device "MyFirstStorageDevice" to read.
*m
10-Jan 21:20 MyBaculaStorage JobId 5: Ready to read from volume
"FullAuto-0001" on file device "MyFirstStorageDevice"
(/usr/local/bacula/volumes).
10-Jan 21:20 MyBaculaStorage JobId 5: Forward spacing Volume
"FullAuto-0001" to file:block 0:216.
10-Jan 21:20 MyBaculaStorage JobId 5: Elapsed time=00:00:01,
Transfer rate=5.177 K Bytes/second
10-Jan 21:20 MyBaculaDirector JobId 5: Bacula MyBaculaDirector 7.0.5 (28Jul14):
Build OS:               amd64-portbld-freebsd9.1 freebsd 9.1-RELEASE-p19
JobId:                  5
Job:                    RestoreFiles.2015-01-10_21.20.07_04
Restore Client:         MyFirstBaculaClient
Start time:             10-Jan-2015 21:20:09
End time:               10-Jan-2015 21:20:09
Files Expected:         8
Files Restored:         8
Bytes Restored:         4,168
Rate:                   0.0 KB/s
FD Errors:              0
```

CONTINUES NEXT PAGE

CONTINUED

```
FD termination status:  OK
SD termination status:  OK
Termination:            Restore OK

10-Jan 21:20   MyBaculaDirector   JobId 5:   Begin pruning Jobs older than 6 months .
10-Jan 21:20   MyBaculaDirector   JobId 5:   No Jobs found to prune.
10-Jan 21:20   MyBaculaDirector   JobId 5:   Begin pruning Files.
10-Jan 21:20   MyBaculaDirector   JobId 5:   No Files found to prune.
10-Jan 21:20   MyBaculaDirector   JobId 5:   End auto prune.

*
```

Done. Boom. Just like that.
Let's go look at them.

```
# cd /tmp/bacula-restores/usr/local/etc/bacula/
# ls
bacula-dir.conf         bacula-fd.conf       bacula-sd.conf      bconsole.conf
client-myclient.conf    filesets.conf        pools.conf          schedules.conf
```

Let's do a diff here:

```
[root@baculatest:/tmp/bacula-restores/usr/local/etc/bacula] # diff . /usr/local/etc/bacula/
Only in /usr/local/etc/bacula/: INSTALLED
diff ./bacula-dir.conf /usr/local/etc/bacula/bacula-dir.conf
53a54,66
>      Job {
>      Name          = "RestoreFiles"
>      Type          = Restore
>      Client        = MyFirstBaculaClient
>      FileSet       = MyFirstFileSet
>      Storage       = MyFirstStorage
>      Schedule      = MyFirstSchedule
>      Messages      = MyMessages
>      Pool          = FullFile
>
>      Where         = /tmp/bacula-restores
>      }
>
Only in /usr/local/etc/bacula/: bacula-dir.conf~
```

That bothered me for a second when I first read it, but yes, that's exactly correct. The **Restore** job was added after the initial backup was run.

But Wait! There's More! Lots, Lots More!

You've just gone through a lot of stuff. There is a lot to learn about Bacula. There are many configuration options and possible choices. I like that. There is no one-way to do things. You can pick and choose.

What I'm keen to try very soon is `bacula-sd` to `bacula-sd` **Job Copy**. I want to store some of my backups at a remote location, without transporting the tapes there.

The rest of this section is various things I thought of while writing this article. I thought they should be included and deserved their own respective sections.

The Retention Periods I Use

I set my **File** and **Job Retention** values very high, higher than my **Volume Retention**. A **Volume** cannot be recycled until all the **Jobs/Files** it contains are pruned from the database (**Catalog**). By keeping **File** and **Job Retention** set to 3 years, I ensure that my **Volumes** will be untouched for at least 3 years or the **Volume Retention** period, whichever is less.

Volumes Might Still Have Your Backups

Even though **File**, **Job**, and **Volume Retentions** have passed, and the **Volume Status** has changed to **Purged**, the backup data remains intact on that **Volume**. It is not until the **Volume** is **Recycled** and new data is written to the **Volume** that your backup data is lost.

This design consideration is part of Bacula's absolute hesitation at overwriting your data. If you absolutely need the backup from a **Purged Volume**, you can get it back using the Bacula utility programs. I recommend trying `bls` and `bextract`.

You should also look into using `Bootstrap Files` with your **Jobs** because of their use with `bextract`.

Backups of Catalog and Configuration

In addition to backing up my **Catalog** dump with Bacula, I copy that dump, all Bacula configuration files, and all `Bootstrap Files` to multiple locations. I do this to avoid ever having to use `bscan` to reconstruct my **Catalog** from my **Volumes**.

Problems?

`bacula-dir` is usually the hardest daemon to get configured and running. Keep an eye on `/var/log/bacula.log` and `/var/log/messages`. If all else fails, try starting `bacula-dir` from the command line, directly:

```
1 /usr/local/sbin/bacula-dir -u bacula -g bacula -v -c\
  /usr/local/etc/bacula/bacula-dir.conf -f -d10
```

Thanks for Reading

The Bacula configuration files I used for this article can be downloaded from http://www.langille.org/examples/bacula7/

Bacula has provided me with a great platform for backing up my systems. It gets used every day. I back up to ZFS file systems which have compression enabled and automatic snapshots. It's fantastic. Every day, those backups are copied to tape. Some tapes get stored at a remote location, just in case. Hopefully I will never have to recover from a real disaster, but I know Bacula has saved me many times when I've deleted/overwitten something.

I hope it serves you well. ●

Dan Langille has been using FreeBSD since 1998 and almost immediately he started documenting his experiences. This online journal eventually became The FreeBSD Diary. He is very good at describing the step-by-step procedures to perform a wide variety of tasks, from changing your prompt to creating and maintaining jails.

Having studied as a software engineer, Dan has picked up sysadmin and database skills along the way. This passion has led into his current DevOps job with the [Cisco Talos Security Intelligence and Research Group] (http://blogs.cisco.com/tag/talos2), where he is doing both coding and sysadmin work, and loving it.

He is the founder of BSDCan, PGCon, FreshPorts, and FreshSource. He's a keen mountain biker and seems to thrive on deadlines.

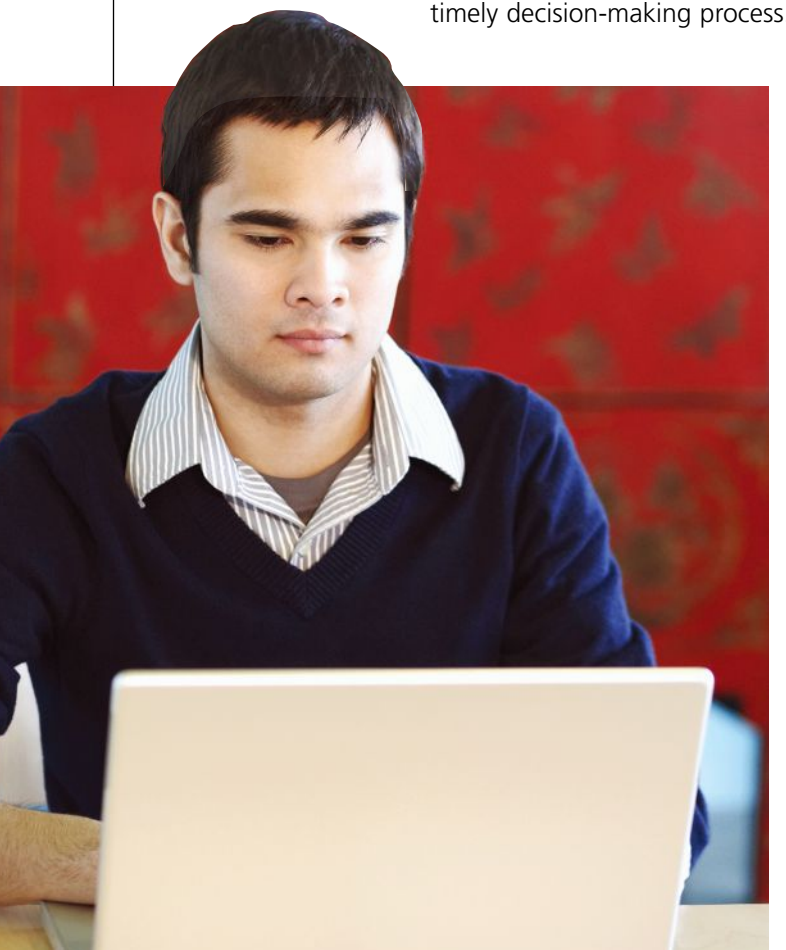
FreeBSD Project

Grant Proposal Submissions

by Ed Maste / The FreeBSD Foundation is a United States-based nonprofit organization dedicated to supporting and promoting the FreeBSD Project and community worldwide. The Foundation fulfills this mission in a number of ways, including sponsoring FreeBSD events and travel, providing legal support, and funding and managing development projects. Development grants are offered to support the development of an identified project or operating system feature.

In this article I'll discuss the process of submitting a grant proposal, and the detail it must contain. Grant proposals are for specific, actionable projects, and include identified developers who will complete the work, as well as cost information. The FreeBSD Foundation is always interested in understanding the FreeBSD community's desires and goals, and desires feedback on prioritizing the project development list. But that feedback is received outside of the grant process.

From time to time the Foundation will make an explicit call for proposals. This allows proposals to be compared and considered as a group and allows for a more efficient and timely decision-making process. A call for proposals may include information on specific areas of focus guided by the Foundation's overall goals. Although we prefer to receive proposals in the context of a call for proposals, we do entertain unsolicited proposals at any time.



Grant proposals may be submitted by an individual developer or a company. The proposal may relate to work on any subsystem or infrastructure within the FreeBSD Operating System or Project. New kernel drivers, userland libraries and tools, third-party software ports, performance optimization, network stack research, documentation, and web-based project tools are all examples of projects the Foundation will consider.

Anyone in the broad FreeBSD community may submit a proposal—proposals are not limited to committers in the FreeBSD Project. However, proposals from those who are not committers will normally identify someone in the FreeBSD Project who will take responsibility for shepherding the work into FreeBSD.

There is no set size for the scope of a project in time or cost. A proposal may describe a project that is a few days of work, or a few months. There are, however, a few general guidelines on areas of focus. Proposals are especially encouraged for work on software, documentation, or infrastructure that is otherwise underserved by the FreeBSD community. That is, the Foundation is particularly interested

in proposals that fill in gaps in either volunteer work or work that is already being funded by others.

Proposed projects should integrate with other projects, goals, and technologies promoted by the Foundation. For example, new daemons and userland tools should consider making use of the Capsicum capability and sandboxing framework. Software development projects must include a description of the associated documentation that will be produced—new or updated documentation is a primary requirement. Reference should also be made to testing, whether unit testing, system testing, or other forms, and in general, new tests should integrate with FreeBSD's ATF/Kyua test framework.

Finally, consider advocacy for the proposed project; one of the Foundation's goals is to make others in the broader open-source and free-software communities aware of the progress being made in FreeBSD. This might take the form of blog postings, talks at BSD-specific and other conferences, or magazine articles on successful projects.

Writing a Grant Proposal

Grant proposals are evaluated on technical merit, cost effectiveness, and the degree of impact the project will have on FreeBSD. A successful proposal will be very clear on the benefits the project will provide. The Foundation's website has full details on the format required for submissions, but some guidelines are presented here.

Title

Choose a brief but memorable title for the proposal. The Foundation will refer to the project in announcements, blog postings, and newsletters. Choosing a good title at the time of project submission promotes consistency in this communication.

Contact Information

Be sure to include your name, a company name if applicable, email and postal address, and phone number.

Abstract

The abstract, or summary, provides a brief overview of the entire project in 50 to 100 words. It should describe the high-level goals, benefits, and approach to be taken. The Foundation may use the abstract in its newsletter or other publications.

Introduction

Longer proposals and those with complex details should have an introduction that provides history or background information on the need for the project. It may be omitted from short and simple proposals.

Project Description

The description should explain in adequate detail what is

to be done, explain why it needs to be done, and why it will provide value to the FreeBSD Project and community. It should demonstrate that the submitter is qualified to do the project and has the resources necessary to complete the project within the stated time and constraints. The description should also include the project deliverables, an overview of the development process, and reference to testing and documentation.

Technical Monitor

A technical monitor oversees the project's progress and ensures that the project's output is of sufficient quality. The monitor is typically a volunteer from the FreeBSD community or a Foundation staff member, but in exceptional cases may be a paid expert.

Project Costing

The proposal must include full, complete costing information, including the developer's fees, any required hardware purchases, reviewer's compensation, and applicable taxes and ancillary fees. If the project has optional components, they may be indicated with separate subtotals, but the base project must remain viable even if none of the optional components are funded.

The Foundation does not have a set guideline on overall project costs or hourly rates, and evaluates each proposal on its own merit. Grants are awarded in support of work that the submitter wishes to take on to improve the FreeBSD Project. As a result, proposed costs are typically not comparable with proprietary or commercial consulting rates.

Timeline and Milestones

The proposal should include a table with a proposed start date, and expected completion dates for each milestone or deliverable. Include per-milestone costing if the project will include interim payments.

Biography

The proposal should include two or three paragraphs describing the submitter's experience and history or relationship within the FreeBSD Project.

Submission Process

The Foundation's website explains the proposal submission process (https://www.freebsdfoundation.org/documents/#Proposals). The full Proposal Submission Guidelines are also available there, including a sample proposal (https://www.freebsdfoundation.org/documents/FreeBSDProposalSubmission.pdf).

A draft proposal may be submitted for initial review and comments, to help refine a final version. Draft proposals must be clearly identified as such.

Once a final proposal is ready it should be submitted as a text or PDF file by email to proposals@freebsdfoundation.org. The FreeBSD Foundation's project committee will review the proposal and reply with its decision.

Project work can begin once a proposal is approved. Brief status updates are to be provided on a weekly basis while the project is in progress, and more detailed interim reports and a final report must accompany each completed milestone.

We look forward to supporting you in improving the FreeBSD Project! ●

Ed Maste manages project development for the FreeBSD Foundation and works in an engineering support role with the University of Cambridge Computer Laboratory. He is also a member of the elected FreeBSD Core Team. Aside from FreeBSD and LLDB, he is a contributor to a number of other open-source projects, including QEMU and Open vSwitch. He lives in Kitchener, Canada, with his wife, Anna, and sons, Pieter and Daniel.

Examples of FreeBSD Foundation Project Grants

- Porting FreeBSD to AArch64, the 64-bit ARMv8 architecture
- PCIe Hot Plug infrastructure
- Opencrypto update with modern AES modes
- Updated system video console
- Multipath TCP for FreeBSD (Master's research project)
- Capsicum application sandbox framework integration
- Superpage support for ARMv7
- New TCP Congestion Control algorithms
- FreeBSD IPv6 stack performance analysis
- Supporting growfs on mounted file systems
- Documentation project infrastructure enhancements

We're excited to announce that March 15, 2015 marks the 15th Anniversary of the FreeBSD Foundation!

You've helped up accomplish so much during the last 15 years and we look forward to continuing that progress through out the rest of 2015. The areas we'd like to invest in include the following:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure
- And More!

Thank you for all of your continued support. We can't do it without you!

Support FreeBSD®

Donate to the Foundation!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system. Our mission is to continue and increase our support and funding to keep FreeBSD at the forefront of operating system technology.

For 15 years, the FreeBSD Foundation has been proudly supporting the FreeBSD Project and community thanks to people like you. We are incredibly grateful for all the support we receive from you and so many individuals and organizations that value FreeBSD.

Make a gift to support our work in 2015. Help us continue and increase our support of the FreeBSD Project and community worldwide!

Making a donation is quick and easy.
Go to **freebsdfoundation.org/donate**

The FreeBSD FOUNDATION

freebsdfoundation.org

FreeBSD Foundation Updates

by Deb Goodkin / **Hello**, *FreeBSD Journal* readers! I'm excited to write our inaugural Foundation Updates column. After publishing this informative and inspiring magazine since January 2014, we've decided we need to keep you informed of what the Foundation does to help FreeBSD, and hence this new column.

Who Are We Anyway?

The FreeBSD Foundation is a 501(c)(3) nonprofit organization dedicated to supporting and promoting the FreeBSD Project and community worldwide. We were founded 15 years ago by Justin Gibbs, our current president, to provide a legal entity to protect the FreeBSD trademark, originally owned by Walnut Creek CDROM. We are now nine board members strong with six full-time staff members. We support funded development projects, promote FreeBSD, provide legal support, bring the community together at conferences and summits, and address needs in underserved areas. You can find out more about our history by reading Justin's article in the March/April 2015 issue of this magazine.

How Have We Helped FreeBSD?

The Foundation's most visible work is project development and funding with half of our budget allocated to this area. With Ed Maste as project development director, the Foundation has been able to increase the number of projects it can support and to make the process more efficient and successful. Over this last year, both development staff and project grant recipients were responsible for a large number of feature improvements and bug fixes.

The Foundation's largest project to date is the FreeBSD/arm64 port, which represents a collaborative development effort where the Foundation facilitates a broader project with multiple participants. Another project is the MPTCP—Multipath TCP implementation for FreeBSD—the subject of a research master's degree at Swinburne University in Melbourne. Currently, the Foundation is also funding the PCIe hot plug project as an individual project grant.

In addition to Ed Maste, who also does code development, we have two other full-time staff members who work on various areas of the operating system. Some of this work includes Secure Boot, the autofs-based automount daemon, dynamically loadable libthr, Intel DMA remapping, and migration to the ELF Tool Chain project tools.

One of the benefits of having long-term permanent staff is the ability to maintain projects and contribute improvements beyond a fixed timeline. Over the last quarter, Foundation staff contributed improvements to the UEFI boot process, vt(4) system console, in-kernel iSCSI stack, the virtual memory subsystem, and many others.

The Foundation's second-largest focus in the last year relates to the promotion of FreeBSD for education. We are supporting a project that is creating university-level curriculum and conference tutorials for teaching operating systems concepts using FreeBSD. The text being used for the course is *The Design and Implementation of the FreeBSD Operating System*, written by three Foundation board members, with course materials developed by Dr. Robert Watson and George Neville-Neil, both board members of the Foundation. The first version of the masters-level class was taught during the Lent term at the University of Cambridge by Dr. Watson, and the

practitioner-focused tutorial was taught by George Neville-Neil over two days at AsiaBSDCon. Other course and tutorial variations are being developed, including expanding the current content for longer, 12-week semesters and full 5-day tutorials.

With a full-time marketing director, Anne Dickison, we have been able to create additional literature with which to promote FreeBSD, which is now disseminated at open-source conferences around the world. Brochure titles include What Is FreeBSD?, How to Get Involved in FreeBSD, FreeBSD 10.x, Writing a FreeBSD Testimonial, and more. We continue to brainstorm new ideas on how to recruit people to the Project and tell the world about FreeBSD, and we look forward to speaking with the community to help further develop the Foundation's materials.

As I mentioned above, the Foundation was originally begun to safeguard the Project's intellectual property. This has grown into providing legal support for the Project, signing license agreements and NDAs to give FreeBSD developers access to proprietary documents, and hardware and software to add pertinent features and functions to FreeBSD.

The addition of systems administrator Glen Barber, a full-time staff member who also supports FreeBSD Release Engineering and Cluster Administration, has helped the Foundation provide high-quality and on-schedule releases. This month, Glen spent several days at the East Coast U.S. colocation facility, generously provided by New York Internet, installing and configuring a number of servers purchased by the FreeBSD Foundation to support the FreeBSD Project.

Supporting Infrastructure

FreeBSD infrastructure is always in need of growth and improvements, and the Foundation helps by purchasing the equipment needed, as well as by helping to install it and providing support. We've spent almost $50,000 this year purchasing servers, cables, memory, hard drives, and other hardware needed to upgrade and improve the FreeBSD infrastructure. The Foundation also covers power costs at some of the facilities. Supporting our team members as well as other FreeBSD contributors in managing these systems guarantees that things are running smoothly for the Project.

Bringing People Together

The Foundation provides opportunities for bringing people together by sponsoring conferences and summits around the world that help facilitate collaboration, learning, and sharing knowledge and experiences.

It's important to bring in new people to foster a diverse community that will help drive the sustainable growth of FreeBSD—new ideas, energy, and passion help propel positive changes in the community. To find these people, promote FreeBSD, and support the community, the Foundation will be sponsoring and/or attending these upcoming events: BSDCan, Ottawa Developer/Vendor Summits, OSCON, vBSDCon, womENcourage, EuroBSDCon, Cambridge Developer Summit, Grace Hopper, OpenZFS, LISA, and the Bay Area Developer/Vendor Summit.

The Foundation was proud to be a platinum sponsor for AsiaBSDCon, which was held in March. Five Foundation team and board members attended the conference, gave talks/tutorials, and interacted with FreeBSD contributors to share their knowledge, work on projects, and learn what others are doing in the community. We also had FreeBSD representatives at USENIX FAST '15, FOSDEM, SCALE, and a few other open-source conferences to promote and share their love of FreeBSD.

Our sponsorship of Grace Hopper in Houston, Texas, and womENcourage 2015 in Uppsala, Sweden, brings FreeBSD awareness to female college students and other women in technology. At womENcourage, a group of FreeBSD representatives will participate in a panel titled Open Source as a Career Path.

Additionally, we fund travel grants for FreeBSD contributors so they can attend these events to work face-to-face with other FreeBSD people. To receive a travel grant, a FreeBSD contributor submits a travel grant application, explaining the importance of a specific conference to their development and to FreeBSD. A committee reviews the applications and will normally respond within two weeks. This year, we're pleased to provide travel grants to 13 FreeBSD contributors to attend BSDCan in June.

Commercial Users Are Shining a Light on FreeBSD

We are actively asking companies to write FreeBSD testimonials that we publish in our monthly newsletters and on our website. Many big-name, commercial users of FreeBSD—such as Network Appliance, Netflix, Verisign, Juniper, WhatsApp, and more—have written testimonials that show how they've been successful using FreeBSD.

By offering companies an opportunity to showcase their use and share their reasons for using FreeBSD, we are helping to shine a positive light on FreeBSD and also reinforcing that FreeBSD is a reliable, well-designed, secure, and stable operating system that other companies can also benefit from.

Providing a Professional Quality Publication

As you already know, *FreeBSD Journal* is a professional-quality, online magazine that publishes important, cutting-edge content. We're pleased to share that not only does the publication now reach over 8,000 subscribers, but that the number of subscribers and advertisers continues to grow, which will help us meet our eventual goal of a self-funding publication. We are proud of the success of the magazine and grateful for the guidance from its editorial board and for editorial contributions from leading authors in the field. It's a great source of helpful and useful information for all Unix-like operating system users.

Going Forward

Going forward, Foundation Updates will highlight the work the Foundation has been doing over the previous two months. This will keep you informed on what it is doing to help the FreeBSD Project and community, and offer an ongoing understanding of how the Foundation spends the Project's money. A high percentage of the donations we receive from readers like you goes directly to helping FreeBSD.

We are proud of the work being done on FreeBSD by volunteers from around the world and are grateful for the opportunity to serve the community. We take our responsibility as stewards of the Project's funds very seriously.

To find out more information about the Foundation and what it does, go to https://www.freebsdfoundation.org. You can also subscribe to the Foundation's monthly e-newsletter by signing up at the bottom of our website.

Finally, as you may know, the Foundation is entirely supported by donations from people like you. Please consider helping us continue the work that we are doing by making a donation and encouraging your colleagues and companies to do the same. Go to https://www.freebsdfoundation.org/donate and make a donation today! We can't do it without you. ●

Deb Goodkin is the Executive Director of the FreeBSD Foundation. She's thrilled to be in her 10th year at the Foundation and is proud of her hard-working and dedicated team. She spent over 20 years in the data storage industry in engineering development, technical sales, and technical marketing. When not working, you'll find her on her road or mountain bike, running, hiking with her dogs, skiing the slopes of Colorado, or reading a good book.

PORTSreport

by Frederic Culot

ACTIVITY on the ports front was at a high level during the March-April period, with important changes brought by our volunteers, especially on package building infrastructure. We also had the pleasure of welcoming one new developer during this period.

NEW PORTS COMMITTERS AND SAFEKEEPING

In the March-April period, only one new committer was granted a ports commit bit, and he is Michael Moll, who is mentored by swills@ and mat@. No commit bits were taken in for safekeeping.

IMPORTANT PORTS UPDATES

Several exp-runs were performed (14 actually) to check whether major ports updates are safe or not. Among those, we can mention the following highlights:

- Xfce updated to 4.12.0
- libtool updated to 2.4.6

Other exp-runs were performed that prevented some updates from being applied as failures were reported. This is the case for the boost library update, which will require more work to reach to ports tree.

As always, please read the /usr/ports/UPDATING file carefully before updating your ports, as manual steps may be involved.

INFRASTRUCTURE UPDATES

A lot of important work to improve the FreeBSD ports building infrastructure is done in the background, and therefore may go unnoticed by end-users. For example, new package builders were added during the last two months to permit three package builds per week instead of just one. Of course, this means more frequent package updates for end-users. The goal is to provide package builds on a daily basis, for which optimization of poudriere (our utility for creating and testing FreeBSD packages, see https://www.freebsd.org/doc/en/books/handbook/ports-poudriere.html) is being worked on. A new tool is also under development to monitor the latest package builds: the results are shown on a dedicated web page (https://pkg-status.freebsd.org). Those who want to participate in the development can find the source code on github (https://github.com/bdrewery/pkg-status.freebsd.org).

STATISTICS

For the two-months of March and April, 4,602 commits were applied to the ports tree, which is an increase of more than 10% compared to the last period. On the bug reports front, 1,324 PRs were closed, which is also a slight increase compared to the January-February period. Let's hope those figures continue to increase and that many more new developers join our ranks in the months to come!

Frederic Culot has worked in the IT industry for the past 10 years. During his spare time he studies business and management and just completed an MBA. Frederic joined FreeBSD in 2010 as a ports committer, and since then has made around 2,000 commits, mentored six new committers, and now assumes the responsibilities of portmgr-secretary.

conference REPORT

by David Maxwell

(https://2015.asiabsdcon. org)

AsiaBSDCon 2015

If you use software, you probably have a favorite program. Since you use it regularly, you know the things you like about it and the things you would like to change. However, with most software, you'll never get attention from the organization that created it, let alone get to interact with the developers who make the tools.

Now, imagine you could go to a place where the developers of amazing software came together and gave presentations about the next great program they're working on, or stories about the process of creating that system you enjoy using. Not only that, but you can interact with them during their talks, or discuss things one-on-one in the hallway, or even get to know them over a meal!

BSD conferences are amazing, informative, and exhausting! From three to five days of talking, listening, debating, and learning will challenge your endurance and your mind as well. From sunup to sundown, you'll encounter many of the brightest people you'll meet in a lifetime.

This year, March 14 and 15 were the main session days for AsiaBSDCon, held in Tokyo, Japan. There were also tutorial days and Developer Summit meetings on the days before the main talks. It was this writer's first time attending AsiaBSDCon.

I enjoy visiting places I've never been to before. I like the challenge of trying to communicate in places where not everyone speaks English. Of course, I got a cell phone data sim card at the airport, so I had ready access to maps and Google Translate, but I did encounter difficulties nonetheless. There is apparently a sign language for counting that has not spread to North America, and holding up two hands, with one finger and 5 fingers respectively means 15, while holding up one finger in the palm of the other, open hand means 6. How that resulted in my order of 30 chicken McNuggets for lunch, I'm still unsure.

The number of people who move through the Tokyo transit systems is amazing. Arriving at Narita Airport, it was easy enough to find the express train to Tokyo central station. Getting out of Tokyo station was less straightforward. After coming up what seemed like five escalators through various track levels, I found that every direction was labelled "exit" of one type or another. Some exits led back down stairs again and up other stairs and I would later discover the inter-train routing could include many stairs and long walks between platforms as well. Eventually I got the hang of it and found a combination of Google Maps and the station's wall map of exits and the local area

worked quite well. After the long flight from Toronto though, I decided to escape and take a cab to my hotel, rather than drag luggage through a foreign-language-labeled maze, filled to the brim with people moving in every direction on their way to work, school, or play. As I would discover the next day, this wouldn't be my only encounter with maps labelled only in Japanese. The Tokyo University of Science conference site also had maps without any English on them.

I have been a user of the NetBSD operating system since 1993, and got my privilege to commit changes to the source code in 2001. I have attended every one of the BSDCan conferences held annually in Ottawa, Canada. Despite my familiarity with the project and many other developers, AsiaBSDCon was still an eye-opening experience.

For me, the conference opened with a NetBSD Developer Summit the day before the main talks. It was followed by a NetBSD Birds of a Feather session in the evening. There were some familiar faces there, and also many developers I had not met in person before. It was fascinating to see how vibrant the NetBSD community is in Japan—there were many presentations on impressive work being done on and with NetBSD. The majority of the presentations were conducted in English, and the language challenge usually only came up during question and answer sessions.

I should mention that while I was not attending any at this conference, there were also several tutorial sessions available on the two days before the main talks. If you want more of an in-depth coverage of a topic, or a thorough introduction to jump-start your learning, these sessions are an excellent option.

The majority of the Summit presentations were live-streamed for those who could not attend in person. A small portion of time was set aside for member-only discussions, which was not streamed.

There is apparently a sign language for counting that has not spread to North America, and holding up two hands, with one finger and 5 fingers respectively means 15, while holding up one finger in the palm of the other, open hand means 6.

• • • • •

How that resulted in my order of 30 chicken McNuggets for lunch, I'm still unsure.

I believe most of the thanks for organizing the event goes to Masanobu Saitoh and Jun Ebihara, though I'm certain others helped in many ways. It's great to hear that Japanese NetBSD developers are winning awards in their region for their diligence in presenting NetBSD talks and demonstration tables at more events than any other open-source project!

All of this was followed by dinner and discussions at a traditional Japanese restaurant; don't forget to leave your shoes at the door. The locals were very helpful in learning how to navigate the native cuisine and eating customs.

The next day was the start of the main conference. I gave a presentation about my tool "pipecut" in the morning, then enjoyed Peter Hessler's presentation about using BGP as a transport for real-time spam blacklists. Apparently Peter has been running this project for two years, but it was the first time I had heard about it. The application of a network routing protocol with the appropriate set of update properties to the problem of distributed blacklisting was great to hear about.

Another memorable talk from the first day was Martin Husemann's presentation on teasing out all of the "ARM processors are little-endian" assumptions in the codebase that had to be overcome to get NetBSD running on the CubieTruck platform. These kinds of talks are great for showing you not to give up when you face a technical challenge—but how to identify the malleable parts of the system that can be adapted to work within the design constraints of the hardware.

I would be remiss in this write-up if I didn't mention Warner Losh's exploding lunchbox! The bento box lunch provided at the conference included a set of instructions about how to partially unwrap them, and then pull a string which sets off a chemical reaction in a puck under the meal, allowing it to heat up to steaming hot right on the table in front of you. Warner's string broke when he pulled it, and he attempted a workaround of opening the (no user maintainable parts) box and quickly pulling on the remaining string. Either the strength of the pull or the disassembly of the box caused the puck to expend all of its available energy in one shot. I'm sorry I didn't take any photos of the show.

Photos left to right: visiting the Yasukuni Shrine; one of several tables at the closing dinner; tables with 25 embedded platforms that could run NetBSD; locating the conference venue.

Keynote presentations were adjoined to the lunch break on each of the two talk days. Dennis Ferguson and K. Robert Elz each presented some of their thoughts and involvement in the evolution of BSD Unix, as an underlying OS for routing at Juniper Networks, and for connectivity and education in Australia, respectively.

The final day of the conference ended with many of the attendees going out for dinner together. There was a group dinner on the first day as well, but I bowed out of that one due to jet lag. There is a lot of laughter to enjoy and a lot of stories to be shared with the people whose names you see on emails, but who you may never have met in person before, or see only once a year at a conference.

I stayed for a few days after the conference to explore Tokyo and Kyoto, as this was my first visit to Japan. I saw many of the Japanese monuments, temples, and shrines, and took in shopping in the famous Akihabara electronics district, and other areas. After enjoying AsiaBSDCon 2015 so much, I hope it will not be my last visit to Tokyo.

Even if you can't find a way to get to a BSD Conference in person, many of the talks are streamed live online, and many of the talks are recorded so you can view them later. Search for the "bsdconferences" user on YouTube, and you'll find a large number of conference session recordings.

If you do come to a BSD Conference, you will probably find the answers to all the hardest questions that have left you stumped, as well as insights into how technology is changing, and how the BSD Operating Systems are adding support and enhancing their frameworks to take advantage of new hardware, new protocols, and new computing paradigms.

Who knows? A year from now, you could be using the latest version of your favorite software—the version that includes the feature you always wanted, your idea, that the developer loved when you met and talked about it at a BSD Conference!

David Maxwell is the Director of Threat Intelligence at eSentire Inc. David has over 25 years of experience as an open-source user and developer, and has been particularly active in the NetBSD community. He currently sits on the advisory board for the BSD Certification Group and the program committee for the annual BSDCan conference. He was also a NetBSD Security Officer 2001–2005, on the NetBSD Foundation Board of Directors 2009–2011, and a contributor to the bestselling O'Reilly title *BSD Hacks* by Dru Lavigne.

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with BSD CERTIFICATION

Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

NEED AN EDGE?

BSD Certification can make all the difference.

Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification

YOU'LL HAVE WHAT IT TAKES!

SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION** can bring you to the attention of companies that need your skills.

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

svn UPDATE

by Glen Barber

THIS EDITION OF SVN UPDATE FEATURES A NUMBER OF VIRTUALIZATION UPDATES AS WELL AS AN IMPRESSIVE UPDATE TO THE XZ(1) COMPRESSION UTILITY.

stable/10@r280632

(https://svnweb.freebsd.org/changeset/base/r280632)

FreeBSD has historically provided excellent ABI/API backwards-compatibility, which makes it possible to run binaries compiled for older FreeBSD versions on newer FreeBSD releases. For example, as long as the kernel is compiled with the COMPAT_FREEBSD4 (to provide the 4.x kernel compatibility layer) that is included in the kernel configuration (and the misc/compat4x port is installed to provide older shared libraries, if needed), it is possible to run a binary compiled for FreeBSD 4.4 on FreeBSD 11-CURRENT.

This also means that it is possible to run a FreeBSD 4.x jail on an 11-CURRENT machine. There is one problem, though. Although the jail userland would recognize it as 4.x, the jail environment would still reflect 11-CURRENT.

Recently, jail(8) was updated to allow tuning the kern.osreldate sysctl(8) for the jail environment so anything that requires knowledge of the kernel version could see a 4.x kernel.

To use more recent FreeBSD versions as examples, if you have a machine running 11-CURRENT with two jails, one for 9.3-RELEASE and one for 10.1-RELEASE, the kern.osreldate for both jails would reflect 11000XX (where XX right now is 73), and you can individually set the jail kernel version to match what the __FreeBSD_version value is (from /usr/src/sys/sys/param.h) for that release.

head@r281316

(https://svnweb.freebsd.org/changeset/base/r281316)

The xz(1) utility is a general-purpose compression utility similar to gzip(1) and bzip2(1), and was updated to the upstream version 5.2.1. One of the impressive features the 5.2.x version provides is multithreading support, allowing use of all available CPU cores on a system when compressing files.

For recent FreeBSD releases, the Release Engineers have provided installation images in both compressed and uncompressed formats. Since multithreading support was added, the amount of time for a set of snapshot builds to complete (from start to finish, for all supported architectures on the branch) has been reduced by a significant amount of time. (As it turned out, the majority of time was spent on compressing the resulting images, and not actually building the FreeBSD sources or even creating the installation medium.) Previously, a full set of FreeBSD 11-CURRENT snapshots would take between 12 and 16 hours, where now it is around 7 hours, which is quite impressive.

When multithreading support was first added, I did a few benchmarks to see how the resulting image size was affected with various threading flags passed to xz(1). The '-T' flag takes a numeric argument, which is the number of cores to use, and the special argument '0' (zero) will default to all available cores on the system.

The FreeBSD release-build machines recently purchased by the FreeBSD Foundation to support the Release Engineering Team are 48-core machines with 128GB RAM. The tests I ran were with '-T 1' (single-threaded), '-T 10' (10-threads), '-T 20' (20 threads), and '-T 0' (in this case, 48 threads). I also ran additional tests using the '-e' xz(1) flag, which in some cases can result in higher compression at the expense of longer time to compress the file.

The disc1.iso image used for this test was 613353472 bytes, just under 585 Mb. The single-threaded test resulted in an image 336.15 Mb in size in 4.5 minutes. The 10-thread test resulted in an image 337.47 Mb in size in 32.55 seconds. The 20-thread test, to my surprise, resulted in an image 337.47 Mb in size as well, in 23.71 seconds. The 48-thread test (using '-T 0') also produced the same 337.47 Mb image size, in 17.37 seconds. This is quite impressive

```
# time xz -T 1 -k disc1.iso ; mv disc1.iso.xz disc1.iso.1.xz
266.534u 1.149s 4:27.68 99.9%   81+193k 4+43029io 0pf+0w

# time xz -T 10 -k disc1.iso; mv disc1.iso.xz disc1.iso.10.xz
256.588u 2.850s 0:32.55 797.0%  81+192k 5+43198io 0pf+0w

# time xz -T 20 -k disc1.iso; mv disc1.iso.xz disc1.iso.20.xz
303.180u 5.393s 0:23.71 1301.4% 81+192k 5+43198io 0pf+0w

# time xz -T 0 -k disc1.iso; mv disc1.iso.xz disc1.iso.48.xz
285.227u 9.958s 0:17.37 1699.3% 81+192k 4+43198io 0pf+0w

# time xz -T 0 -e -k disc1.iso ; mv disc1.iso.xz disc1.iso.48e.xz
345.370u 8.275s 0:20.34 1738.6% 81+192k 4+43187io 0pf+0w

# ls -al disc1.iso*
-rw-r--r--  1 root  wheel  613353472 Feb 20 19:50 disc1.iso
-rw-r--r--  1 root  wheel  352488020 Feb 20 19:50 disc1.iso.1.xz
-rw-r--r--  1 root  wheel  353870676 Feb 20 19:50 disc1.iso.10.xz
-rw-r--r--  1 root  wheel  353870676 Feb 20 19:50 disc1.iso.20.xz
-rw-r--r--  1 root  wheel  353870676 Feb 20 19:50 disc1.iso.48.xz
-rw-r--r--  1 root  wheel  353786156 Feb 20 19:50 disc1.iso.48e.xz
```

indeed. The test with '-T 0' and '-e' produced an image 337.39 Mb in size in 20.34 seconds, so did not provide much benefit for the additional three seconds.

For completeness, the time(1) output and directory listing are included above.

For large file compression, this xz(1) update has certainly proved to be quite significant.

head@r280259.

(https://svnweb.freebsd.org/changeset/base/r280259)

The FreeBSD Foundation, in collaboration with ARM, Cavium, and Semihalf sp.j., is sponsoring FreeBSD developer Andrew Turner to port FreeBSD to the 64-bit ARM platform (aka., aarch64).

The initial support for FreeBSD/aarch64 was added to 11-CURRENT in r280259, and has been continuously worked on since. The goal for this project is to bring FreeBSD/aarch64 to Tier-1 support status, including release installation medium and third-party packages.

Initial support for producing FreeBSD/aarch64 release medium was added to 11-CURRENT in r281802, which work was sponsored by the FreeBSD Foundation. At present, virtual machine images for use with the Qemu emulator, as well as memory stick installation images, are available on the FreeBSD FTP mirrors.

Please note though that a Qemu EFI loader file is needed to boot the virtual machine images. For additional details (as well as the checksums for the EFI loader file), see the snapshots announcement mailing list archives, which include an example of how to boot the images (https://lists.freebsd.org/pipermail/freebsd-snapshots/2015-May/000147.html).

Thank you, dear readers, for supporting the FreeBSD community, the *FreeBSD Journal,* and of course, The FreeBSD Foundation.

Don't forget, development ISO and preinstalled virtual machine images (in VHD, VMDK, QCOW2, and RAW formats) of the FreeBSD-CURRENT and FreeBSD-STABLE branches can be found on the FTP mirrors, and are built weekly: ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/.

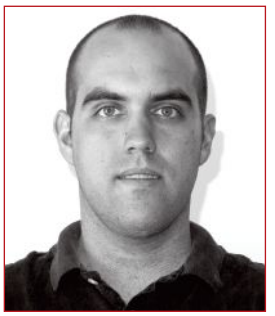
As always, development snapshots are not intended for production use; however, we do encourage regular testing so we can make the next FreeBSD releases as great as you expect them to be. ●

As a hobbyist, **Glen Barber** became heavily involved with the FreeBSD project around 2007. Since then, he has been involved with various functions, and his latest roles have allowed him to focus on systems administration and release engineering in the Project. Glen lives in Pennsylvania, USA.

this month In FreeBSD

BY DRU LAVIGNE

In the last issue, we took a look at the FreeBSD Project's participation in Google Summer of Code. This month, I had the opportunity to interview **Julio Merino**, a former Google Summer of Code student, who now works at Google. His journey from a student who participated in open source is an interesting one that spans multiple BSD projects. His interest in automated testing and code review has provided benefits to both his career and the projects in which he is still involved.



Julio, tell us a bit about yourself. What is your technology background, how did you become interested in automated testing and code review, and what attracted you to the FreeBSD Project?

I have been around computers pretty much since I was born. My father bought an Amstrad CPC 6128 around that time to keep himself busy and taught me the basics of programming. From there I went on to learn lower-level programming languages and alternative operating systems to Windows, passing by OS/2, Linux and, finally, the BSDs around the year 2000. Along the way I got a degree in computer science and then a master's. I now work for Google, and am a FreeBSD and NetBSD committer.

I became interested in automated testing after working on my first Google Summer of Code (GSoC) project in 2005. At that time, I developed a new file system (tmpfs) for NetBSD and, as I was following an exploratory approach, I needed scripts to validate that the code worked and that I did not introduce regressions with every change. Writing the tests themselves was interesting, but it was frustrating to see no infrastructure to glue them together. From there, I went on to write ATF (the Automated Testing Framework) as another GSoC project in 2007.

Since I joined Google in 2009, I have written tons of tests and have been exposed to better testing practices. This, combined with my previous interest in purely functional programming idioms, has taught me a lot about how to properly organize code so that it can be easily tested. In 2010, I concluded that the tools shipped by ATF could not grow to cover my testing goals, and so I started work on Kyua.

The code review process has a similar story. I have been around open source for many years (since 1997 or so) and have contributed to various projects. There were code reviews in the form of mailing list patches, but I never thought the process was "right." On the other hand, Google is known for requiring code reviews for every change, so the last few years have gotten me to learn a lot about this. While annoying (literally) at first, formal code reviews are incredibly valuable. Code reviews will not catch every single problem, but depending on how well you pick your reviewers, you'll be able to avoid embarrassing mistakes and potentially major design flaws. In fact, when I write open-source code of my own, I now feel "naked" without code reviews validating what I'm doing.

• *You have been involved for some time with creating and implementing testing frameworks, first with ATF and then with Kyua. For the benefit of users who are new to testing frameworks, please provide an overview of the motivations for and benefits of using testing frameworks and what is involved for integrating those frameworks into large, existing software projects.*

At this point, ATF and Kyua are complementary. Kyua is a "testing framework for infrastructure software." What this means is that Kyua offers a runtime engine to run unit and integration tests and provides the

mechanisms to collate their results. Kyua does not intend to provide a Continuous Integration (CI) system, although that's something that was originally in my plans. There are perfectly valid solutions out there that do this today—see Jenkins or Travis CI. Kyua attempts to fill the gap in the system, which is providing the mechanism by which test programs are run from these CI systems.

I think Kyua's major strength is its modular system with seamless support for various test program types. This is a key feature in permitting the integration of legacy test programs into a new test suite with minimum effort, without having to rewrite them to use the flavor-of-the-day testing library.

ATF is a set of libraries to write test programs in a variety of languages. ATF supports C, C++, and shell, and all test programs implemented using ATF offer a consistent user interface among one other. If you happen to be an atf-sh user, I'd like to suggest you look at shtk's unittest module, which is my latest attempt at implementing a more modern testing library for the shell that follows the common xUnit idioms. And if you use atf-c++, consider googletest as an alternative, a much more mature C++ testing library.

• *You were also instrumental in the process of integrating Phabricator into FreeBSD. Why is code review so important? How difficult was it to integrate from both a technology and a cultural perspective? Do you have any anecdotes on how the Project has benefitted from the code review process?*

Haha, I think calling me instrumental in integrating Phabricator into FreeBSD is a "bit" of an overstatement! Really, all I did was show excitement for the new tool. I was an early adopter and I wrote one of my most popular blog posts evangelizing this new (to some) practice (http://julipedia.meroh.net/2014/05/code-review-culture-meets-freebsd.html).

Code review is important because nobody writes perfect code. And, in particular, because developers (myself included) are usually overconfident when it comes to assessing changes to parts of the code they are not familiar with. It is in those cases where a second up-front look from an "expert" is valuable.

From the technical aspect, I was glad to see that there are reasonable tool implementations in the open-source ecosystem to support this. I wouldn't say integrating Phabricator into FreeBSD's use case was super difficult—although I honestly do not know because I do not administer the FreeBSD machines.

The trickiness of integrating code reviews into an existing community are political. First, because code review is initially seen as a burden by those that have never been exposed to it, and, second, because some developers do not believe in code reviews at all. All said, I think FreeBSD has done a pretty good job at overcoming these obstacles. FreeBSD's Phabricator instance is at 2,200+ reviews at the time of this writing, which clocks at about 6 reviews per day. Considering this was only deployed a year ago, in my book that counts as success!

• *What is the current state of automated testing within the FreeBSD Project? Where do you see the FreeBSD Project five years from now with regard to testing and code review?*

FreeBSD is in a reasonable spot with regard to testing. We have a toolchain that can build and run tests. We have continuous execution of those tests via Jenkins. These are all important.

But there is a very long way from where we are to the desire of being able to run tests on each commit, and to restrict such testing to only the tests that are impacted by the commit's scope. Solving this problem is incredibly difficult though, if only because make(1) is not a particularly nice build system to track dependencies at this high semantic level. Not to say that the additional amount of resources we'd need would be significant.

Another aspect that deserves improvement is kernel-level testing. There currently is no way in FreeBSD to run noninvasive—that is, unprivileged and without side effects—tests for kernel-level code. In my opinion, this is something that NetBSD has gotten right with its adoption of rumpkernels (https://wiki.netbsd.org/rumpkernel/). I think implementing this approach in FreeBSD would be a major advance.

• *Do you have any suggestions for readers who do not have a FreeBSD src commit bit, but who are interested in assisting with testing?*

Simple: contribute tests! Tests do not have to be complex to be useful, nor do they need to exercise obscure parts of the system. We could go a long way by just having simple integration tests for every single command-line utility and unit tests for all public library functions. Even the most simple test ends up failing at some point during the development process. The

reason these are so important is because they help in preventing regressions. If a developer breaks a test, even the most simple one, he'll be forced to reason about the breakage and decide whether the test needs to be fixed/updated or whether his original change was actually wrong. (The answer to this question is not always easy!)

Another way to help is to refactor the code base for testability. Writing testable code is something that requires up-front planning/knowledge, and lots of code in the BSDs was not written with tests in mind. A lot of the code has existed for many years, and automated continuous unit testing was not a hot topic back then.

• Do you have any other projects up your sleeve?

"Not really at the moment. My free time is scarce with a 3-year-old around the house and a baby arriving soon. Plus, I like writing technical essays as well, and those take a good amount of time.

So whenever time permits, or when I can remind myself to use my 20% "free time" at Google, I use such time for Kyua. My plan is to get Kyua 1.0 out the door this year and I've been really hard at work at this—which is kind of a problem, because I've been focusing exclusively on the coding aspects of adding parallel support to Kyua and ignoring all related email. Apologies if you have been affected by this! •

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.

THE FOLLOWING RESOURCES PROVIDE MORE INFORMATION ABOUT THE TOPICS DISCUSSED IN THIS INTERVIEW:

- https://wiki.freebsd.org/TestSuite
- https://wiki.freebsd.org/CodeReview
- https://github.com/jmmv/kyua
- https://github.com/jmmv/shtk
- https://code.google.com/p/googletest/
- http://julipedia.meroh.net/

THROUGH OCTOBER

BY DRU LAVIGNE

2015 Events Calendar

The following BSD-related conferences will take place in the next 4 months. More information about these events, as well as local user group meetings, can be found at **www.bsdevents.org**.

OSCON • July 21–23 • Portland, OR

http://www.oscon.com • There will be a FreeBSD booth in the Expo Hall of OSCON. Registration is required for this event.

Texas LinuxFest • Aug. 21 & 22 • San Marcos, TX

http://2015.texaslinuxfest.org/ • There will be a FreeBSD booth at TLF, to be held in the San Marcos Convention Center. The BSDNow crew will be in attendance, and the BSD certification exam will also be available during this event. There is a nominal registration fee for this conference.

vBSDcon • Sept 11–13 • Reston, VA

http://vBSDcon.com • The second biennial vBSDCon conference will take place at the Sheraton in Reston. This event begins with a one-day FreeBSD Developers Summit, followed by two days of presentations and BOF sessions. Registration is required for this event.

womENcourage • Sept 24–26 • Upsulla, Sweden

http://womencourage.acm.org/ The aim of this conference is networking and exploring career opportunities for women in computer science and related disciplines. Both men and women are welcome at this event. Several members of the FreeBSD Project will be participating on a panel about the benefits of open source and one will be giving a workshop on getting started with open source. There will also be a FreeBSD booth in the career fair. Registration is required for this event.

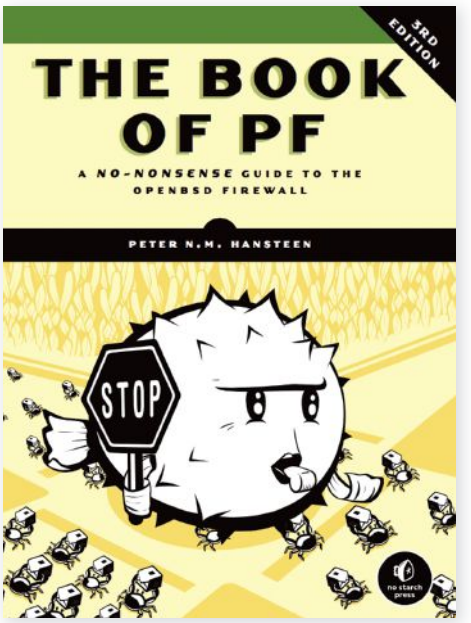
EuroBSDCon • Oct 1–4 • Stockholm, Sweden

https://2015.eurobsdcon.org/ • The primary European BSD conference will take place this year at Stockholm University. There will be a two-day Developer Summit, two days of tutorials, and two days of presentations. Registration is required for this event. Paul Vixie will deliver this year's keynote.

Are you aware of a conference, event, or happening that might be of interest to *FreeBSD Journal* readers? Submit calendar entries to editor@freebsdjournal.com.

BOOKreview

by Steven Kreuzer



The Book of PF:
A No-Nonsense Guide to the OpenBSD Firewall

By Peter N.M. Hansteen

• No Starch Press, 3rd edition (October 18, 2014)
• ISBN: 159-3275897 • 248 pages

From time to time I get asked why I prefer to use FreeBSD and what advantages it offers over Linux. One of my many reasons is that it ships with a modern, high-performance firewall that is under active development by security-focused developers. Since its release in 2001, OpenBSD's PF packet filter has provided systems and network administrators with a well-thought-out, stateful firewall with insanely simple syntax that makes it easy to write and debug rules. If it's not obvious by now, I am a huge fan of PF and therefore my review of *The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall* may come off as a bit biased. However, if you have never worked with PF, but have had the misfortune of using iptables on Linux, you will quickly understand the origin of my bias.

Peter N. M. Hansteen's *The Book of PF* is indeed a "no-nonsense guide to the OpenBSD firewall," covering everything from the configuration basics all the way up to building highly available and redundant networks. Now in its 3rd edition, it has been completely revised to cover the most up-to-date developments in OpenBSD 5.6, FreeBSD 10.0, and NetBSD 6.1. The book has also been expanded to include an in-depth tour of the new priorities and queues system that has replaced the older ALTQ system in OpenBSD 5.5.

The first two chapters are aimed at the relative newcomer and focus on getting you up to speed with PF. Only expecting the reader to have some basic Unix administration experience alongside some minor knowledge of networking with TCP/IP, the material in this book is presented in a tutorial style that makes the content very accessible and easy to follow. From the third chapter onward, the book covers several different deployment scenarios ranging from running a simple firewall to protect your workstation all the way up to building a high-availability network that could rival commercial offerings costing hundreds of thousands of dollars.

One of the things I loved about this book that I rarely see in technical books is that after the brief introduction, Hansteen wastes no time getting down to brass tacks. Are you one of the unlucky souls who still needs to run an FTP server? Here is what you need to do. Do you want some separation between hosts on your office LAN and those connected to the Internet? Here is a recipe you can tweak to fit your needs. Everything is straight to the point with no fluff or pages wasted explaining how to compile a piece of software and seven different Linux distributions that seems all too common today. The book is a fairly quick read at only 248 pages, but my guess is that your copy will become dog-eared pretty quickly because you will find yourself rereading chapters and constantly going back to it as a reference.

One of the things I really enjoy about PF and what Hansteen does a great job of conveying is that PF is a nice foundation upon which to build simple and effective solutions using other pieces of software that greatly expand the utility of PF. *The Book of PF* has an entire chapter dedicated to "proactive defense," which provides simple exam-

ples of how to set up an adaptive firewall to deal with brute force attacks on your listening services. If you have ever had a machine running SSH connected directly to the Internet, I am sure you have watched countless brute force attempts clog up your log files. Numerous solutions exist that attempt to address these problems, but most of the time they require you to run some poorly documented Perl script as root which parses your log file and adds rules to block offending IP addresses. One of the reasons I love PF is that it has a vast array of additional tools with which it seamlessly integrates to deal with these nuisances in a very elegant way.

Perhaps you run your own mail server, and day in and day out see countless connections from machines attempting to relay spam. Dealing with spam is almost as old as email itself and countless efforts have been proposed with varying degrees of success. Analyzing the content of a message or checking to see if the sender has been added to a blacklist only seems to work for a short period of time before the spammer gets wise and changes the methodology. You find yourself in a never-ending arms race, which is why OpenBSD developers focusing on fighting this battle at the network level use PF and spamd. Hansteen spends a great deal of time covering various methods whereby you ruin a spammer's day and even have a little fun at their expense by responding with 1 byte per second before rejecting their messages. Even if you don't run a mail server, setting up spamd and watching spammers get stuck for several hours waiting for EHLO is incredibly satisfying.

Once you deploy PF into production, it's always a good idea to keep a close eye on what's happening. Depending on how complex your setup is, you could end up generating tons of data that becomes a monumental task to sift through. Fortunately the author dedicated an entire chapter to logging, monitoring, and collecting statistics. Not only does it cover the built-in methods to analyze what is going on, but it also covers some third-party tools that can be installed from the ports tree and provide additional real-time insight and historical trends that allow you to keep an eye on your network.

When writing this book, I believe Hansteen's intention was for you to have a copy of it next to you while you tinker, and the book wraps up by helping you get your configuration just right before you deploy it into production. While out-of-the-box PF defaults to having a sane config that should cover typical deployments, he offers practical advice on some of the knobs you could tweak and others that you maybe should leave alone. Even though throughout the entire book Hansteen offers advice and techniques to help build networks that are possible for a mere mortal to debug, the final chapter deep-dives into additional techniques on how you can test and debug your rule set, which can come in handy when things just don't seem to be working as you would expect.

While the book mainly focuses on the feature set of PF in OpenBSD, when necessary, the author does a very thorough job at describing the differences you will encounter when using PF on FreeBSD and NetBSD. The examples provided are based on real life problems you might have to solve at some point, which makes it very easy to follow and gives you a nice foundation to build upon when actually deploying PF into production.

Hansteen's stated goal of *The Book of PF* is to "help you build the network you need" by starting off with a little theory and slowly working his way up to more and more complex configurations, all the while offering plenty of examples on how to control a packet as it attempts to make its way through your network. I believe that not only did he achieve this goal, he exceeded it, and I would highly recommend this book to any systems or network administrator. This book will save time and frustration by walking you through commonly encountered pitfalls and if you are using PF on any platform, you should have a copy of this book in your library. If you happen to have friends who currently battle with iptables day in and day out, this book would make a wonderful gift for them, just so you can let them know what they are missing. ●

> **"ONE OF THE THINGS I REALLY ENJOY ABOUT PF AND WHAT HANSTEEN DOES A GREAT JOB OF CONVEYING IS THAT PF IS A NICE FOUNDATION UPON WHICH TO BUILD SIMPLE AND EFFECTIVE SOLUTIONS USING OTHER PIECES OF SOFTWARE THAT GREATLY EXPAND THE UTILITY OF PF."**

Steven Kreuzer is a FreeBSD developer and Unix systems administrator with an interest in retrocomputing and air-cooled Volkswagens. He lives in Queens, New York, with his wife and dog.

Testers, Systems Administrators, Authors, Advocates, *and of course* **Programmers** *to join any of our diverse teams.*

COME JOIN THE PROJECT THAT MAKES THE INTERNET GO!

★ DOWNLOAD OUR SOFTWARE ★

http://www.freebsd.org/where.html

★ JOIN OUR MAILING LISTS ★

http://www.freebsd.org/community/mailinglists.html?

★ ATTEND A CONFERENCE ★

- *http://www.**vBSDcon.com**/*
- *https://2015.**Eurobsdcon.org**/*

The FreeBSD Project

The FreeBSD FOUNDATION